




University
of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Computer Science	Spring semester, 2019 Open/ Confidential
Author: Jungwon Seo	 (signature of author)
Faculty supervisor: Tomasz Wiktorski	
Title of Master's thesis: Minimum Word Error Rate Training for Speech Separation	
Credits: 30 ECTS	
Keywords: Cocktail Party Problem, Speech Separation, Source Separation, Deep Learning, CNN, RNN, DNN, WER	Number of pages: 106 + supplemental material/other: - Code included as link in Appendix Stavanger, June 15 2019



Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Minimum Word Error Rate Training for Speech Separation

Master's Thesis in Computer Science
by

Jungwon Seo

Internal Supervisor

Tomasz Wiktorski

June 14, 2019

“It is not enough to do your best; you must know what to do, and then do your best. ”

W. Edwards Deming

Abstract

The cocktail party problem, also known as a single-channel multi-talker problem, is a significant challenge to enhance the performance of automatic speech recognition (ASR) systems. Most existing speech separation model only concerns the signal-level performance, i.e., source-to-distortion ratio (SDR), via their cost/loss function, not a transcription-level performance. However, transcription-level measurement, such as word error rate (WER) is the ultimate measurement that can be used in the performance of ASR. Therefore we propose a new loss function that can directly consider both signal and transcription level performance with integrating both speech separation and speech recognition system. Moreover, we suggest the generalized integration architecture that can be applied to any combination of speech recognition/separation system regardless of their system environment.

In this thesis, first, we review the techniques from the primary signal processing knowledge to deep learning techniques and introduce the detailed target and challenge in speech separation problem. Moreover, we analyze the several famous speech separation models derived from a deep learning approach. Then we introduce the new loss function with our detailed system architecture, including the step-by-step process from pre-processing to evaluation.

We improve the performance of the existing model using our training approach. Our solution enhances average SDR from 0.10dB to 4.09dB and average WER from 92.7% to 55.7% using LibriSpeech dataset.

Acknowledgements

First of all, I would like to thank Professor Tomasz Wiktorski for his trust when I first proposed this topic. Despite the lack of progress in the early days, I was able to complete this study successfully because of the advice I received from each meeting.

I am also honored to be with Julie Høvik Aase, my companion, who supports me in all aspects of life in Norway.

Contents

Abstract	vi
Acknowledgements	viii
Abbreviations	xiii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Definition	2
1.3 Usecases/Examples	3
1.4 Challenges	3
1.5 Contributions	3
1.6 Outline	4
2 Literature Background	5
2.1 Automatic Speech Recognition (ASR) systems	5
2.2 Speech Separation: Cocktail Party Problem	6
2.3 Background in Audio Processing	6
2.3.1 Audio Representation	6
2.3.2 Short Time Fourier Transform	9
2.3.3 Soft Masks	10
2.4 Performance Measurements	12
2.4.1 Signal-level Performance Measurements	12
2.4.2 Transcription-level Performance Measurements	13
2.5 Deep Neural Networks	14
2.5.1 Logistic Regression	14
2.5.2 Cost/Loss Function	16
2.5.3 Neuron and Artificial Neural Networks	19
2.5.4 Feed Forward Neural Networks (FNNs)	21
2.5.5 Activation Function	24
2.5.6 Recurrent Neural Networks (RNNs)	25
2.5.7 Convolutional Neural Networks (CNNs)	27
2.5.8 Embedding	32

3	Related Works	35
3.1	Conventional Techniques for Speech Separation	35
3.2	Deep Learning in Speech Separation	35
3.2.1	Understanding the Task	35
3.2.2	Label Permutation Problem	38
3.2.3	Deep Clustering	38
3.2.4	Deep Attractor Network	40
3.2.5	Permutation Invariant Training (PIT)	41
3.3	Inspirational Works and Theory	43
3.3.1	Joint Training with speech recognition system	43
3.3.2	Minimum Word Error Rate Training	45
3.3.3	Reweighting the loss function	46
4	Solution Approach	49
4.1	Introduction	49
4.2	Existing Approaches/Baselines	49
4.2.1	Ideal Ratio Mask	49
4.2.2	VoiceFilter	50
4.3	Proposed Solution	53
4.3.1	Performance Based Weighted Loss Function	53
4.3.2	Generalized Pipelining with Speech Recognition System	54
4.3.3	System Architecture	56
5	Experimental Evaluation	59
5.1	Experimental Setup and Data Set	59
5.1.1	Dataset	59
5.1.2	Preprocessing	60
5.1.3	Generating Audio Mixture	61
5.1.4	Generating new ground-truth transcripts	62
5.1.5	Software/Hardware and Libraries Specification	63
5.1.6	Hyperparameters and System Configuration	63
5.1.7	Training and Testing Policy	64
5.2	Experimental Results	64
5.2.1	SDR and WER Comparison	64
5.2.2	Performance by Different Number of Words	66
6	Discussion	67
6.1	Performance Analysis	67
6.1.1	Evaluation Measurement Analysis	67
6.1.2	Relationship between SDR and WER	68
6.2	Limitations	69
6.2.1	Dataset	69
6.2.2	Training Time	70
7	Conclusion and Future Directions	73

List of Figures	73
List of Tables	77
A Implementation and Demo	79
B Spectrogram Results	81
C Training Process	83
Bibliography	85

Abbreviations

ASR	A utomatic S peech R ecognition
CNNs	C onvolutional N eural N etworks
DPCL	D ee P C Lustering
FNNs	F eedforwd N eural N etworks
RNNs	R ecurrent N eural N etworks
SDR	S ource to D istortion R atio
WER	W ord E rror R ate
PIT	P emutation I nvariant T raining
IRM	I deal R atio M ask

Chapter 1

Introduction

1.1 Motivation

Automatic speech recognition (ASR) is one of the most popular research areas in computers science in past years. As people always have developed new inventions for a more comfortable life, ASR is also believed as one of the keys to guide us the next generation of communication between human and machine. Since speech is the most natural and convenient method for humans to communicate, it is reasonable to apply this method to make us interact with personal devices such as smartphone, laptop, and tablet.

Even though the ASR technology has shown human level accuracy these days; the performance is still not stable in a real-world environment. The main reason is, the ASR system is developed based on the clean audio source. However, there are few chances to meet the noise-free circumstance in our daily life; therefore, environmental obstacles such as noise sound and other's speech should be handled first. Thus, speech separation is considered as essential to escalating the performance of the ASR system.

Speech separation task, which is also called a 'cocktail party problem,' has been tackled from signal level traditional method to spectrogram level deep learning. In the speech separation task, there are various assumptions about the condition of sound. For instance, the target sound can be interfered by background noise, e.g., cars, birds, and wind. Also, it can be interfered by other speakers. When there is more than one speaker, sometimes the number of the speaker is offered, sometimes it is not. Therefore, the machine learning task can be supervised learning task and unsupervised learning task based on prior knowledge.

In this paper, we focus on the supervised learning situation, which means that the model knows the number of speakers, furthermore, reference audio (same speaker's different speech) is also offered. Based on the existing system, we suggest the new loss function and system architecture that can improve the performance of its original model.

1.2 Problem Definition

In the speech separation task, many researchers use a signal-level loss to train the model. The main limitation of this approach is, the loss function does not directly represent the performance of the system. In other words, there is a miss match between loss and performance. For instance, many signal-level training systems use mean squared error (MSE) as its loss function to compare the element-wise level difference between original audio and separate audio [1]. However, this way of calculating the loss may miss some criterion since it does not fully cover all the performance measurements that are widely used. Usually, signal-level measurements such as source-to-Distortion ratio (SDR), source-to-Interference ratio (SIR) and source-to-artifact ratios (SAR) are used to evaluate the speech separation system's performance, and MSE performs reasonably for those measurements.

However, transcription-level performances such as word error rate (WER) and character error rate (CER) are considered the core measurement in the end. Because the primary purpose of speech separation is to enhance the accuracy of speech recognition, therefore, even though the separation process achieves high SDR, it does not necessarily mean that WER is also better. In other words, while the model is training, the performance may already reach a reasonable point in terms of signal. However, since the loss function does not directly represent its performance, the model may be over-trained. Furthermore, the direction that decreases the loss value does not necessarily mean to minimize the error rates mentioned above. Therefore, the model may converge in unsatisfiable point based on its loss function.

Thus, in this paper, we propose a new way to define a loss function that can directly enhance the WER of the speech separation system. Additionally, we introduce the system architecture to integrate both speech separation and speech recognition system regardless of their system environment.

1.3 Usecases/Examples

The speech separation system can also be called, speech enhancement or voice filter system because the main idea is generating better speech sound from the noisy sound. As personal devices are provided widely, the importance of digital assistant is also rising. Many devices have its own voice assistant system such as Siri from Apple, Bixbi from Samsung and Alexa from Amazon. The speech recognition performance has been improved drastically; however, it is still limited by many environmental conditions such as a noisy situation. Therefore, it is unavoidable to enhance the user's speech quality regardless of its environmental factors. In that manner, our speech separation has tremendous potential for the ultimate voice assistant.

1.4 Challenges

There are two challenges that we mainly focus on. First of all, proposing new loss function that can consider both signal-level and transcription-level is the main challenge. The loss function should be reasonable with sufficient theoretical evidence. Secondly, integrating two systems (speech separation and speech recognition) with minimizing the training performance is also our primary task. Moreover, since each system depends on its own system configuration, it is vital to structure the environment-free integration.

1.5 Contributions

In this paper, we investigate the research area of speech separation from scratch to state-of-the-art technology and propose a new technique that can directly improve the word error rate.

First, we review the speech separation area, including audio processing itself and conventional blind speech separation techniques comprehensively. Secondly, we introduce the standard evaluation measurement for both signal-level and transcription-level. Thirdly, we briefly review the deep learning technology and the way to apply to a speech separation system, moreover, analyze the role of each network' in the separation system architecture.

In the end, we integrate two systems to suggest our solution. The main challenge is to relate the loss function and WER directly through the training process. Therefore, we integrate speech recognition system next to the speech separation process. We choose 'Deep speech' introduced [2] as a speech recognition system, and for the speech separation system, we use 'Voicefilter' from [3]. Since the purpose of the thesis is, experimenting

with the effect of transcription-level loss for the speech separation over the spectrogram-level loss, we fix the performance of the speech recognition system using pre-trained model. A detailed description of those two systems will be introduced later. Furthermore, our system integration approach is generalized regardless of the system environment. Therefore our proposed system architecture can be applied to any other combination of speech recognition and separation system.

1.6 Outline

The paper consists of seven chapters, including this introduction. In chapter 1, we introduce our task and define the problem. In chapter 2, we cover the background literature for both signal processing and deep learning. However, since the range can be varied, we mainly focus on the theory and techniques that are highly relevant to our topic. In chapter 3, we review the speech separation task using deep learning techniques with the existing system; additionally, we introduce some research that can theoretically support our solution. In chapter 4, the model that we are using as a reference is deeply analyzed, and we propose our solution. In chapter 5, the experiment process and result are described, and we discuss our result in chapter 6. Finally, we conclude the result and suggest the future direction for further improvement in chapter 7.

Chapter 2

Literature Background

2.1 Automatic Speech Recognition (ASR) systems

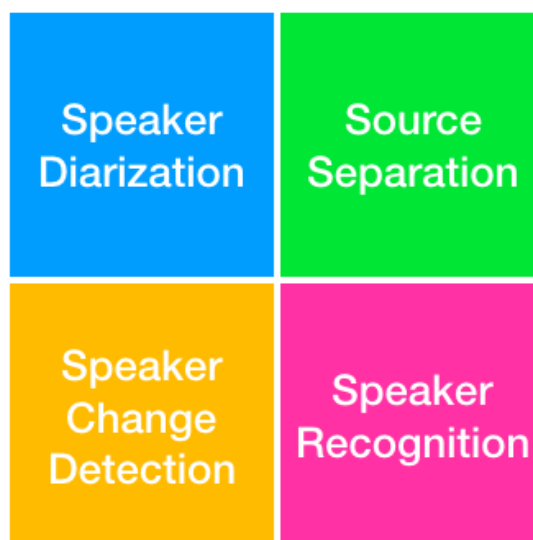


Figure 2.1: Subdomains of ASR

There are four subdomains to improve the ASR system's performance, as shown in Fig.2.1. The **speaker diarization** is the mechanism to find “*Who Spoke When?*”. The **speaker change detection** sounds similar to speaker diarization; however, is the process to mainly focus on the moment that different speaker starts to talk. The **speaker recognition** is to identify the speaker; in other words, it is highly related to the voice print. Lastly, **speech separation**, which is our main topic, takes one part of these preprocessing parts. Those four mechanisms are a vital component to build the outperforming ASR system regardless of the performance of a speech recognition system itself. Moreover, these four mechanisms are intertwined; therefore, it will improve the

overall performance strongly if we consider and implement all the domains. However, in this thesis, we only focus on the speech separation process.

2.2 Speech Separation: Cocktail Party Problem

Speech separation problem is also known as ‘Cocktail Party Problem’ which is named initially from [4] in 1953. As we can notice from the name, it demonstrates the situation where many sounds are existing concurrently in natural auditory environments. The cocktail party problem is a task to extract or separate the target sound from mixed or noisy sounds. The synonym of the cocktail party problem is ‘cocktail party effect’ which describes the human ability that we can concentrate on only the sound that we are interested in. Based on [5], when human intend to focus on one speaker, the auditory system restores the representation of target speaker while suppressing irrelevant noisy speech. However, when the task moves to the computer, this starts to be a challenge.

To adapt human behavior into the computer, two challenges should be considered [6]. First challenge is, as we mentioned all along, how to separate sounds from the mixed sounds. Humans can concentrate on one or two target sound. However, since the computer has uncomparable multi-tasking ability, in the last case, it should be possible to separate all the sounds from the mixed version. Secondly, it is essential to trace and hold the target speaker, especially in conversation situation or long speech, and as we describe above in 2.1, this is also related to other subdomains.

In our project, we focus on the first challenge, and the way to tackle this problem from the perspective of the computer is described in Chapter .

2.3 Background in Audio Processing

Before we dive into audio processing in deep learning, it is essential to understand the fundamental signal processing theory and concept. In this section, we will briefly explore the characteristics of sound and its representation.

2.3.1 Audio Representation

There are two main domains in audio data, often used in most deep learning technology. One is the time domain, and the other one is the frequency domain information. If we represent the audio wave using only the time domain with its amplitude envelope of a varying waveform level of a sound wave over time, it can be shown as in Fig.2.2.

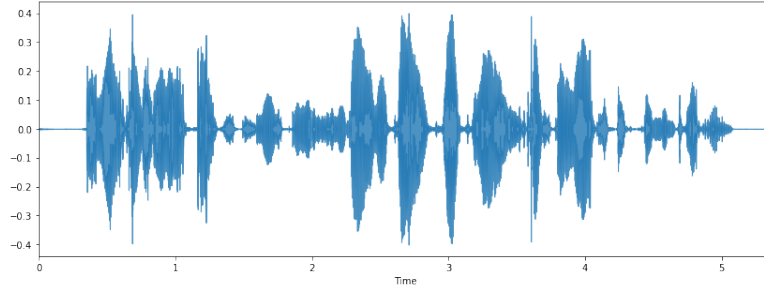


Figure 2.2: Audio wave plot

It is unchallenging to detect ‘when-to-speak’ from the wave plot; however, this information is missing essential details. For instance, since the amplitude represents the loudness of the audio at that moment, the same magnitude does not necessarily mean the same sound. In other words, the amplitude-time domain cannot distinguish between two different speech. Therefore, time-amplitude data miss essential information, and time-amplitude only data is challenging to be used in speech audio.

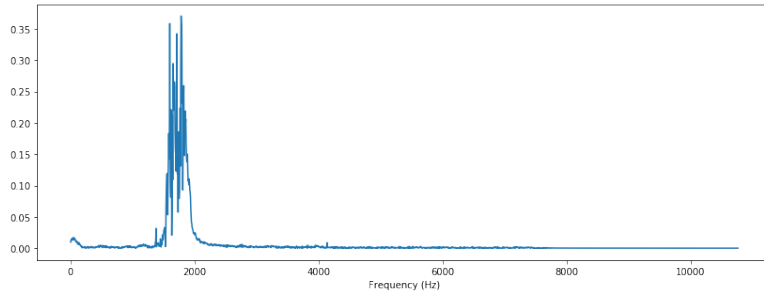


Figure 2.3: Frequency plot

Therefore, many research use frequency domains as core information to differentiate the characteristic of sound. Discrete Fourier Transform (DFT) is widely used to extract the frequency data from the signal. As shown in Fig.2.3, we can plot the amplitude-frequency plot, and we can efficiently distribute the signal based on its frequency.

$$\begin{aligned}
 X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \\
 &= \sum_{n=0}^{N-1} x_n \cdot [\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)],
 \end{aligned} \tag{2.1}$$

To highlight the effect of using the frequency domain, we test a simple example. We test single word audio files from the same speaker. One word is ‘eight’ and the other word is ‘right’. The reason that we use the same speaker’s speech is since male and female have a different frequency range of their ordinary utterance. Therefore sometimes, the same gender mixture is considered as another level of challenge in speech separation task [7].

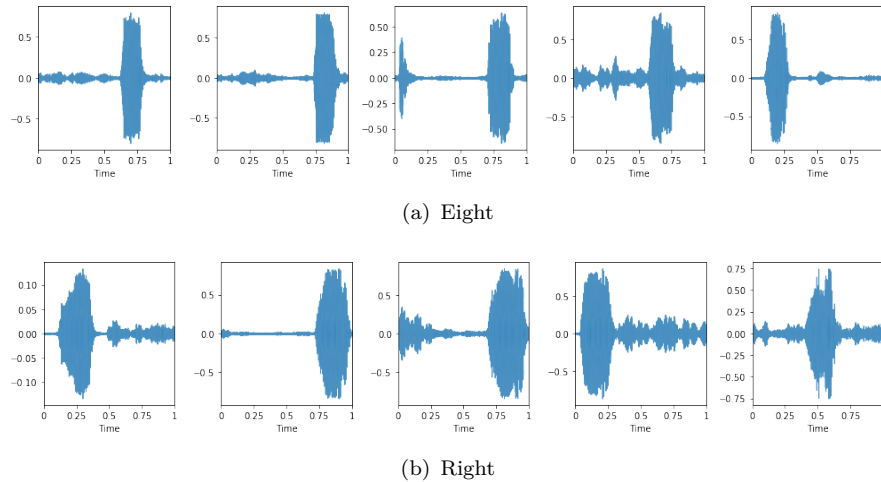


Figure 2.4: Amplitude-Time plot

First of all, the Amplitude-Time plot shown in Fig.2.4, does not show the consistent shape from the same word. Therefore, some graph may look similar even if they are a different word.

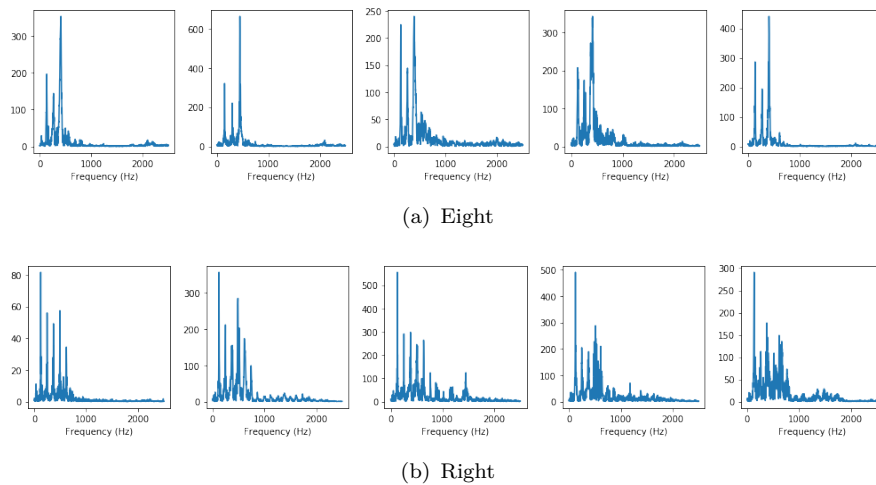


Figure 2.5: Amplitude-Frequency plot

On the other hand, in the Amplitude-Frequency plot shown in Fig.2.5, we can observe some patterns. First of all, the same word shows the consistent pattern, which means, even though the timing of speaking is different in the end they use a similar frequency to sound the same word or character. Secondly, it shows a clear difference between ‘eight’ and ‘right’. If we analyze it visually, in ‘eight’ utterances, the highest pick is normally third or fourth one in the plot. However, in the ‘right’ utterances the highest peak is the first one. We can summarize as ‘right’ has more low-frequency sound than ‘eight’.

However, frequency only information also has a limitation. Since DFT loses the time domain, this information is not helpful when the speech is longer than a certain length.

Moreover, since speech is not a stationary signal, the time domain should be considered simultaneously [8].

2.3.2 Short Time Fourier Transform

Many researchers use the Short Time Fourier Transform (STFT) to obtain both time and frequency domain. Especially when we use deep learning techniques, we use an STFT spectrogram as a front-end in ASR.

$$X(m, \omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \quad (2.2)$$

The Eq.2.2, shows the way to obtain both time and frequency domain using STFT. Where

$$\begin{aligned} x[n] &= \text{input signal at time } n \\ w[n-m] &= \text{length } m \text{ window function (e.g., Hamming)} \\ X(m, \omega) &= \text{DTFT of windowed data centered about time } m. \end{aligned}$$

If we increase m , the window function w will move to the right; then we compute Fourier transform for the frame $x[n]w[n-m]$. In other words, STFT is the repeated Fourier transform using Eq.2.1 for each frame that is shifted by the window function. Therefore, $X(m, \omega)$ obtain both time and frequency domain [9].

To visualize the signal source, we use spectrogram, which is the spectrum of the frequency of the signal aligned by time. Although we use the spectrogram only to confirm the overall progress in this paper, the spectrogram can be the direct input source for some speech research area. Because the machine learning task can be transformed into image processing and it is more researched area than sound processing [10–12].

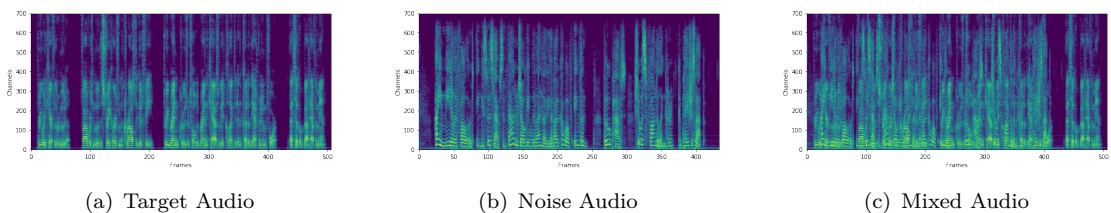


Figure 2.6: Spectrogram after STFT of audio sources

In Fig.2.6, three different examples of STFT spectrogram are presented. The mixed audio 2.6(c), which we will discuss the detail later, is generated by element-wise additive between

target audio 2.6(a) and noise audio 2.6(b). And the color represents the magnitude of each Time-Frequency bin while dark color means silent.

2.3.3 Soft Masks

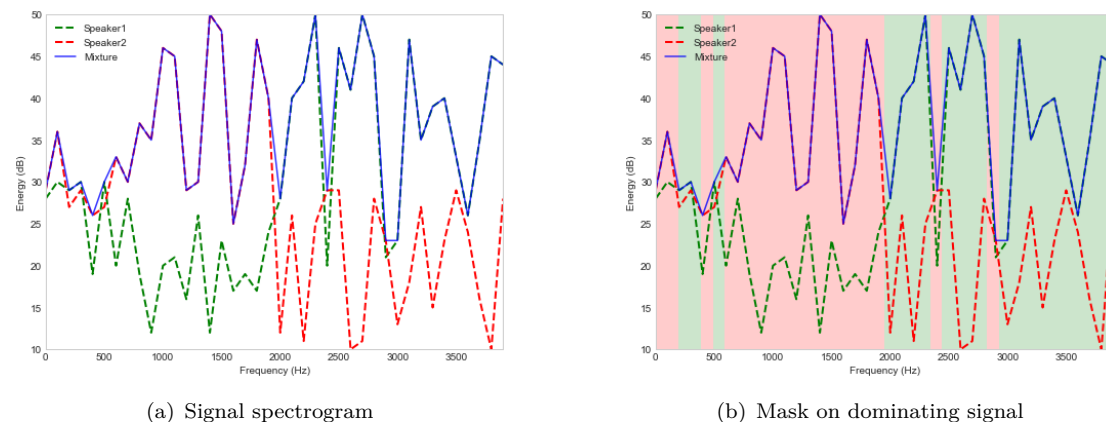


Figure 2.7: Signal separation in the spectrogram

Before we describe the mask concept, we prepare a simple example that can intuitively help to understand the needs of mask in speech separation. In Fig.2.7, we demonstrate the situation when two speech is mixed. The original audio is represented as a dashed line (red and green), and both speakers have different frequency zone since one is female and the other is male. The blue showed the frequency pattern of mixture audio, the notable symptom in this figure is when we mix the audio the information of lower amplitude speech is becoming obsolete. Therefore, it can generate the mask, as shown in the right image, we can separate the corresponding frequency information. And this is the main observation of using soft mask in speech separation [13]. Note that this is a DFT plot, not the STFT plot; therefore, there is no time information.

In order to reconstruct the specific source from the mixed version, many studies use soft mask method [14, 15]. The ideal mask can be generated under the circumstance that all the sources such as target, noise, and mixed are known. There are several versions of masks [16–19], and the most basic mask is binary mask [20].

As denoted in Eq.2.3, the ideal binary mask (IBM) can be generated by comparing between target and mixed source. If the signal-to-noise ratio (SNR) is higher than a certain threshold, it assumes that the T-F unit is derived from the original target source. Therefore, that unit will be 1, and the others will be 0.

$$IBM = \begin{cases} 1 & \text{if } SNR(t, f) > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \quad (2.3)$$

Now, if we compute the element-wise multiplication between the noise source and IBM, then only corresponding T-F bin will remain, and this will be considered as an original sound. If we visualize the IBM of Fig.2.6, it can be shown as Fig.2.8.

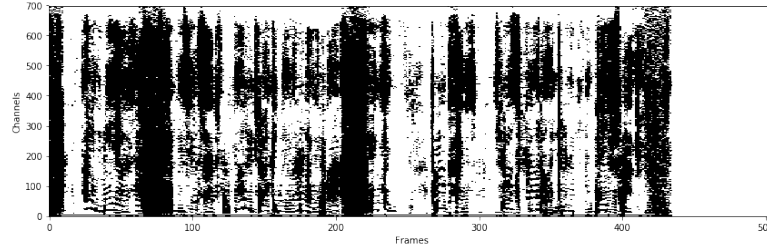


Figure 2.8: IBM Spectrogram

Although IBM is the simplest way to generate mask and its reasonably effective, IBM has significant limitation in terms of speech quality and intelligibility [21, 22]. Especially when the noise is another speech, not the background noise or stationary sound, the performance degrades drastically [23].

To overcome the limitation of IBM, many research starts to use the ideal ratio mask (IRM) as their soft mask [17]. As shown in Fig.2.4, the IRM can be generated by element-wise division between target source ($S(t, f)$) and mixed source ($Y(t, f)$). The constraint of IRM is $0 \leq IRM(t, f) \leq \infty$, however, the majority of the T-F units are in the range of $0 \leq IRM(t, f) \leq 1$ based on many experiments [6]. Therefore, it is possible to estimate the IRM using several activation functions such as softmax, sigmoid, and ReLU at the end of deep neural network.

$$IRM(t, f) = \frac{|S(t, f)|}{|Y(t, f)|} \quad (2.4)$$

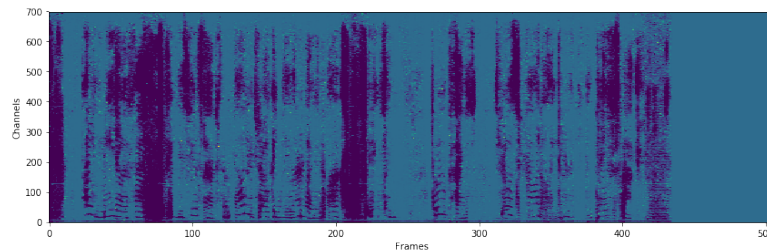


Figure 2.9: IRM Spectrogram

Unlike IBM, IRM uses T-F units assigned the ratio of energy between target and mixed sources; it is more robust against the reverberant condition. The spectrogram in Fig.2.9 shows the IRM from the same sources used for IBM above.

There are variations of IRM, called Ideal Amplitude Mask (IAM) and Spectral Magnitude Mask (SMM). However, in this paper, we stick with the IRM that we describe above.

Although IRM shows the excellent performance to reconstruct the original source from the mixed version, it does not consider the difference of phase between each source. Therefore, phase sensitive mask (PSM) is proposed by [18, 24]. PSM is an extended version of IRM. In Eq. 2.5, $\cos\theta$ denotes the phase difference between target and mixed sources. Furthermore, it has been proven that the PSM leads to higher SNR and is able to generate better clean audio from the mixed audio than IRM [18].

$$PSM(t, f) = \frac{|S(t, f)|}{|Y(t, f)|} \cos\theta \quad (2.5)$$

Although PSM is known to have better performance than IRM, we decide to use IRM due to its simplicity in terms of implementation.

2.4 Performance Measurements

In speech separation, there are several types of measurements that are widely used, and they can be divided into two main categories. First of all, there are signal-level measurements to evaluate performance, especially for speech separation task.

2.4.1 Signal-level Performance Measurements

Traditionally, to compare the quality of the signal when it is exposed to the noise, SNR is widely used. However, to diversify the measurement of the performance, several performance criteria have been introduced for speech separation in [25], more precisely, measurements for the blind audio source separation (BASS).

Before we describe the measurements, there are several terms that we are using as following:

- $S_{estimated}$: Estimated source
- S_{true} : Original source
- $S_{interfere}$: Noise source due to mis-separation
- $S_{artifcat}$: Noise source due to the reconstruction algorithm itself.

The estimated source after the separation can be rewritten as shown in Eq.2.6.

$$S_{estimated} = S_{true} + S_{interfere} + S_{artifact} \quad (2.6)$$

The measurements for the performance evaluation can be written as following three equations[2.7, 2.8, 2.9].

$$SDR = 10\log_{10} \frac{\|S_{true}\|^2}{\|S_{interfere} + S_{artifact}\|^2} \quad (2.7)$$

First of all, **source-to-distortion ratio** (SDR) is the most common metric to evaluate the source separation system, and it consists of the source from both the original and estimated version. It is as an energy ratio between the energy of original audio and the energy of errors from mixed audio and artifacts in decibel. Therefore, higher SDR means better performance.

$$SIR := 10\log_{10} \frac{\|S_{true}\|^2}{\|S_{interfere}\|^2} \quad (2.8)$$

$$SAR := 10\log_{10} \frac{\|S_{true} + S_{interfere}\|^2}{\|S_{artifact}\|^2} \quad (2.9)$$

Source to interferences ratio (SIR) is the energy ratio between the original source and the non-original source, in this case, the mixing audio. **Source to artifact ratio** (SAR) is the ratio between the original source and artifact.

Although all the measurements are meaningful in the different types of audio data, in our task, we mainly focus on SDR to compare easily with the other research.

2.4.2 Transcription-level Performance Measurements

While the measurements mentioned above represent signal-level, there are also transcription-level metrics that should be measured. The Word Error Rate (WER) which is derived from the Levenshtein distance [26] can be denoted as shown in Eq.2.10.

$$WER = \frac{\text{Insertion} + \text{Deletion} + \text{Substitution}}{\text{Number of words}} \quad (2.10)$$

For example, the WER between the sentences below is $WER = \frac{1+1+2}{9} = 0.44$.

- The quick brown fox jumps over the lazy dog.
- The quick red fox jump over the duck's house.

Also, there is a similar measurement to WER called character error rate (CER). CER uses the same equation as WER except for the fact that it uses the number of characters.

However, these transcription-level measurements cannot be the absolute standard when we evaluate the performance of speech separation task, because WER can be varied based on the performance of the speech recognition system. Therefore, the WER will be used to evaluate the relative difference or improvement during the experiment. The detailed usage is described in Chapter 5.

2.5 Deep Neural Networks

To understand the fundamental of deep learning technology and how it can be applied to the signal based data, it is crucial to comprehend underlying network architectures. In this section, we will describe several different models related to the neural networks.

2.5.1 Logistic Regression

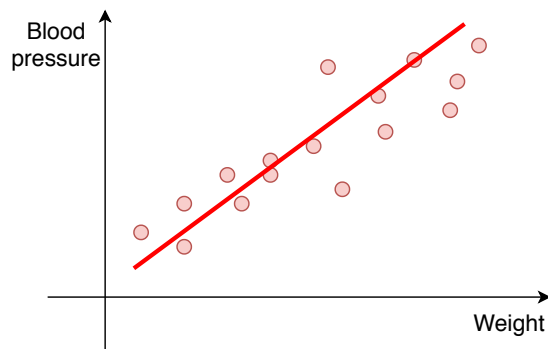


Figure 2.10: Sample graph between blood pressure and weight

Linear regression is the method to estimate the optimal regression coefficient that can represent the relationship between the numerical input variable and the dependent variable the most. For example, if we plot the blood pressure based on several features such as age, height, and weight of an individual people. With known data, we can make a linear equation that shows the relationship between input variables (in this case weight) and blood pressure as shown in Fig.2.10. Later if we assign the unseen data with

same input features we can estimate the possible blood pressure. If we formulate this relationship it can be written as in Eq.2.11.

$$y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots + \beta_px_p + \epsilon \quad (2.11)$$

However, if we use the same approach to classify the potential heart attack, the graph will be like Fig.2.11. Therefore, it is not suitable to use linear regression for the categorical dependent variable. This is why logistic regression is suggested as a classifying algorithm.

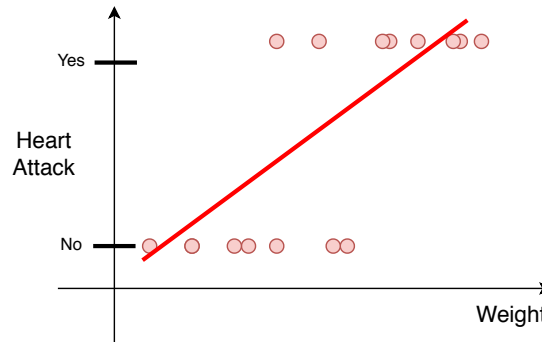


Figure 2.11: Sample graph between heart attack and weight

The core concept in logistic regression is how to represent the non-linear relation as a function. Because many real-world phenomena are following S-curve not linear, **sigmoid** function is proposed ($y = \frac{1}{1+e^{-x}}$), also shown in Fig.2.12.

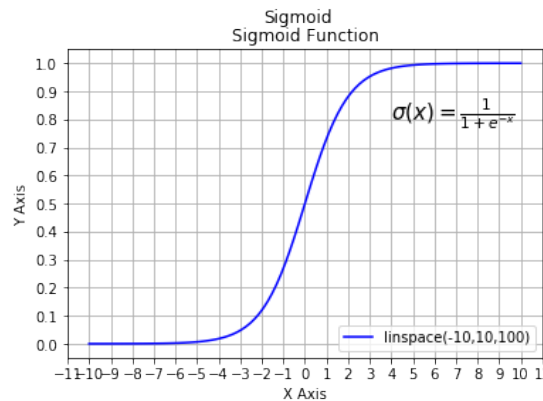


Figure 2.12: Sigmoid Function

To derive the logistic regression mathematically, first, we need to define odds, which are the ratio of the probability that an event A occurs compared to A will not occur.

$$\text{odds} = \frac{P(A)}{P(A^c)} = \frac{P(A)}{1 - P(A)} \quad (2.12)$$

Now we adjust the output range to $[-\infty, \infty]$ when the input range is $[0, 1]$ with assigning logarithm on Eq.2.12 then we can formulate the function as Eq.2.13 called **logit**

$$\text{logit}(p) = \ln \frac{P(A)}{1 - P(A)} \quad (2.13)$$

In logistic regression, the result of the logit transformation is the same as the linear function of x ; therefore, the equation can be written

$$\text{logit}(p) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon \quad (2.14)$$

$$\ln \frac{p_i}{1 - p_i} = \beta \cdot X_i \quad (2.15)$$

Thus, the probability that a dependent variable belongs to a category of 1 with given the particular independent variable x we want to find is

$$p_i = \text{logit}^{-1}(\beta \cdot X_i) = \frac{1}{1 + e^{-\beta \cdot X_i}} \quad (2.16)$$

Now we finally derive a sigmoid function. And this is highly related to the neural networks fundamental model that we will describe later in Sec.2.5.3.

2.5.2 Cost/Loss Function

After we decide the model either linear or logistic regression for a particular task, the following question is, “*How can we find the optimal line that can represent the relationship between input and output the most?*”. Therefore, we need a mathematical formula called loss/cost function to estimate the error between our prediction and real value.

For linear regression, it is simple to build the loss function. If we formulate the linear regression function as $h_\theta(x_i) = \theta_1 x + \theta_0$, also called hypothesis, the main task is finding the optimal θ_1 and θ_0 . Then we only need to average the difference between the estimated value and real value, as shown in Eq.2.17. If the loss is lower, it means our hypothesis represents the relationship better.

$$J(\theta_1, \theta_2) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x_i) - y_i)^2 \quad (2.17)$$

In order to find the minimum loss value from $J(\theta_1, \theta_2)$, a straightforward way is calculating all the possible combination and finding the combination of θ_1 and θ_2 that minimize the

cost. However, this approach is computationally expensive, especially in modern machine learning that requires tons of data.

Therefore, we need to calculate the minimum loss using a more mathematical and computational approach called gradient descent algorithm. It uses the derivative characteristic of a function, in other words, if we derivative a function from a certain point (e.g., $\theta_1 = 1, \theta_2 = 1$), we can find the gradient from that point so that we can estimate the next move that can reduce the further loss. Fig.2.13 shows a simple example.

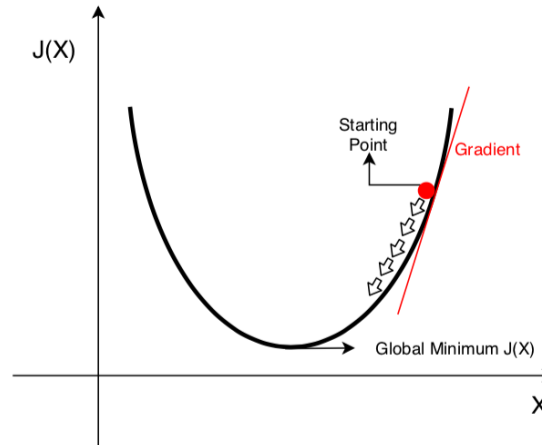


Figure 2.13: Gradient descent simple example

Since there are two parameters (θ_0, θ_1) , partial derivation will be applied for each parameter, and the process will be repeated until it reaches convergence with the following equation:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (2.18)$$

where $j = 0, i = 1$, and α denote the learning rate which decides the distance for the next iteration. Therefore, if we apply the derivation to the loss function that we introduce in Eq.2.17, we can formulate the following equation in Eq.2.19.

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{2m} \sum_{i=1}^m (\theta_1 x^i + \theta_0 - y^i)^2 \quad (2.19)$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} \frac{1}{2m} \sum_{i=1}^m (\theta_1 x^i + \theta_0 - y^i)^2 \quad (2.20)$$

Note that it is essential to update the gradient descent simultaneously when we actually code; otherwise, each parameter's derivation will affect the other parameter's gradient calculation. Therefore, the final pseudocode can be written as Alg.2.1.

Algorithm 2.1 Gradient Descent Simultaneous Update

```

while Repeat until convergence do
  temp 0  $\leftarrow \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$ 
  temp 1  $\leftarrow \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$ 
   $\theta_0 \leftarrow$  temp 0
   $\theta_1 \leftarrow$  temp 1
end while

```

One crucial factor is that cost function should not have a local minimum and should be able to reach the global minimum no matter where the algorithm starts. This type of function is called a convex function.

There are several types of loss function for the target of the model. Mainly we can categorize the loss function into two types: regression and classification task.

For the regression task, again, mean squared error (MSE) is the most common loss function.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \quad (2.21)$$

The alternative loss function is mean absolute error (MAE). Unlike MSE, MAE uses the absolute difference between the prediction and real value.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |Y_i - \hat{Y}_i| \quad (2.22)$$

Lastly, mean bias error (MBE) is the loss function that can consider the direction as well since it sums the signed difference.

$$\text{MBE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i) \quad (2.23)$$

Each loss function has its own purpose; therefore, we can not naively compare that which one is better than the others. For example, even though MSE is the most widely-used one, when the difference should be considered accurately, such as in finance, we can argue that MAE is more suitable loss function.

For the classification task, the most well-known loss function is cross entropy. In the binary classification task, the loss can be formulated as shown in 2.24, where p and y denote the probability and the corresponding label.

$$\text{cross-entropy} = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{if } y = 0 \end{cases} \quad (2.24)$$

$$= -(y \log(p) + (1-y) \log(1-p)) \quad (2.25)$$

If the task is the multiclass classification, then the function can be generalized as shown in Eq.2.26.

$$\text{cross-entropy} = -\sum_{i=1}^N y_i \log(p_i) \quad (2.26)$$

We have covered several loss functions in this section. From the next section, we introduce neural networks concepts and analyze the gradient descent in neural networks.

2.5.3 Neuron and Artificial Neural Networks

The reason that we cover the logistic regression first before we describe the neural networks is if we add the sigmoid function to each cell in the fully connected layer, that one cell is the same as the logistic regression. Therefore logistic regression can be seen as a fundamental of neural networks.

Artificial Neural Networks are inspired by the way the human brain activates the neuron when human during the learning process. It is widely known that the original Artificial Neural Network (ANN) is proposed in 1943 [27]. The structural difference between ANN and logistic regression is whether it has a hidden layer or not. The motivation of the hidden layer is to resolve the huge parameter size issue when the model tries to use more features as its input to deal with non-linearity. For example, in logistic regression when the classes are not linearly separable, it takes the additional feature to obtain the non-linearity hypothesis after combined several features (e.g., $x_1, x_2 \Rightarrow x_1x_2$). However, this approach will increase the parameter size quadratically. Moreover, if we add more cases, the computational cost will keep increasing.

Also, one of the necessary processing for machine learning is feature selection. Usually, human chooses several essential features based on statistical analysis and intuition. However, it is challenging to discover the ultimate combination of feature if the number of input features starts to be considered, such as image data. Hence, ANN solves these issues with a combination of more layers and more neurons.

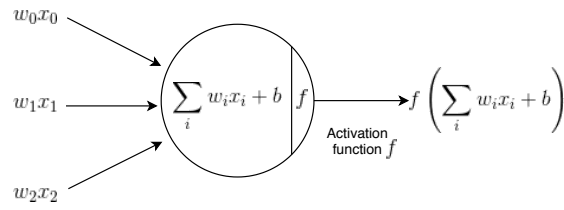


Figure 2.14: Neural network cell with activation function

Fig.2.14 shows the simple ANNs with three input features. If we formulate this, the final output can be written

$$\text{output} = f(w_0x_0 + w_1x_1 + w_2x_2 + b) \quad (2.27)$$

where f denotes activation function, and w_i represents the weight of corresponding features, also is updated during the training process.

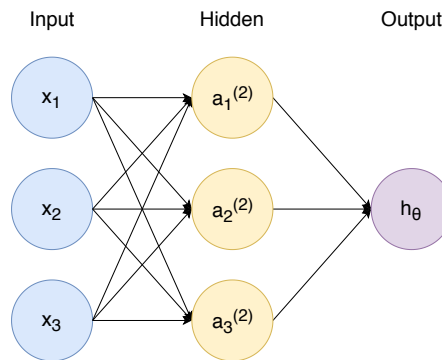


Figure 2.15: One hidden layer neural networks

If we formulate a bit more complex ANNs as shown in Fig.2.15, which contains one hidden layer and three nodes inside, and the other configuration is the same. The superscript on the activation value a_n^l denotes the layer number l (counting from the input layer), and the subscript denotes the node number n .

If we mathematically represent the neural networks, first of all, each output from hidden nodes is:

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \quad (2.28)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \quad (2.29)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \quad (2.30)$$

where $\Theta_{ij}^{(l-1)}$ denotes the weight for each corresponding input feature (i =input node number, j =hidden node number).

Now the final hypothesis ($h_{\Theta}(x)$) can be written as :

$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}). \quad (2.31)$$

Finally, here we introduce the way that ANNs solve the massive feature problem to obtain non-linearity that we mentioned at the beginning of this section.

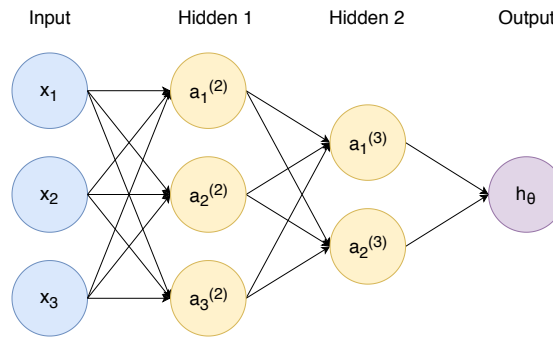


Figure 2.16: Two hidden layer neural networks

Fig.2.16 shows two hidden layers ANNs, and we formulate additional nodes in the same way.

$$a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}) \quad (2.32)$$

$$a_2^{(3)} = g(\Theta_{20}^{(2)} a_0^{(2)} + \Theta_{21}^{(2)} a_1^{(2)} + \Theta_{22}^{(2)} a_2^{(2)} + \Theta_{23}^{(2)} a_3^{(2)}) \quad (2.33)$$

$$h_{\Theta}(x) = a_1^{(4)} = g(\Theta_{10}^{(3)} a_0^{(3)} + \Theta_{11}^{(3)} a_1^{(3)} + \Theta_{12}^{(3)} a_2^{(3)}) \quad (2.34)$$

If we apply the sigmoid function and expand the hypothesis,

$$h_{\Theta}(x) = \frac{1}{1 + e^{-\frac{1}{1+e^{-\frac{1}{1+e^{-T}}}}}}} \quad (2.35)$$

where T denotes simplified term for all the specific parameters mentioned above. Even though, we do not fully expand the all the notation due to its complexity, however, as we can see from the recursive exponential part all it is complex enough to obtain non-linearity and deeper network with more nodes will guarantee to handle more complex non-linearity.

2.5.4 Feed Forward Neural Networks (FNNs)

Now, the rising question is “**What is deep learning?**”. In terms of networks’ architecture, when there is more than one hidden layer (*deep enough*), we call the ANNs as deep

neural networks (DNNs). We have proven mathematically about the effect of deeper networks in Sec.2.5.3, we can also understand more intuitively with the following image recognition example.

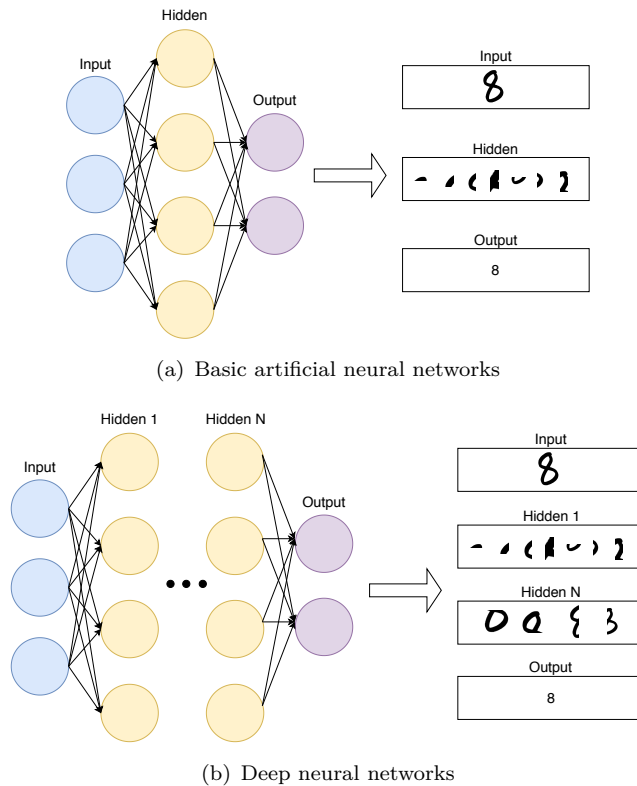


Figure 2.17: Basic architectures of artificial neural networks and deep neural networks

Again, the main difference between basic ANNs and DNNs is the number of hidden layers; in other words, the capacity for understanding the pattern. We visualize the simple handwriting digit image recognition process in Fig.2.17. The DNNs can recognize a more complicated pattern of an image by passing features to hidden layers, while one hidden layer only ANN can recognize the partial pattern of the original digit image.

Since the DNNs contains more logits than one logistic regression, minimizing the loss is also approached differently. First, the process of calculating the value from the input layer to the output layer called forward propagation. Now the same as the previous gradient descent algorithm mentioned above, DNNs also needs to derivate the corresponding function to find the gradient descent, and this process is called **back propagation**.

If we formulate the loss function using cross-entropy for neural networks,

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_{\Theta}(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2 \quad (2.36)$$

where the first half, is the generalized (K classes) term of logistic regression's cross-entropy and the second half is the regularization term which is used to avoid the overfitting with highlighting some features.

Now the task is finding the Θ that minimizes the $J(\Theta)$ the most. However, unlike the simple logistic regression, neural networks require more partial derivation since it consists of many parameters that we have shown in the previous section. Moreover, each parameter Θ is formulated by previous layers Θ ; therefore, the calculation should be chained. This is one of the reasons that DNNs starts to have popularity much later, unlike their original concepts introduced several decades ago (CNN: 1989 [28], backpropagation: 1986 [29]). However, thanks to hardware improvement (e.g., GPU) and convincing algorithm such as backpropagation, DNNs became the most powerful machine learning model now.

The **backpropagation** algorithm is used to calculate the gradient descent efficiently in ANNs. Unlike logistic regression, the gradient descent cannot be calculated at once. If we observe the Eq.2.32 again, the activation a_l values in a certain layer are decided by the previous layer's activation values recursively. Therefore, backpropagation uses the chain rule to calculate the gradient descent.

In ANNs, the training process is following two steps.

1. **Forward propagation:** feed the input features through the network and decide the output. Then, calculate the error between the target value and the estimated value.
2. **Backpropagation:** backpropagate the error to each node in neural networks.

Now, backpropagation is processed. For each node, we can calculate the error (δ_j^l) of node j in layer l , e.g., $\delta_j^4 = a_j^4 - y_j$. Then compute, $\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} \cdot * g'(z^{(3)})$, $\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} \cdot * g'(z^{(2)})$, where $\Theta^{(l)}$ is the vector of parameters for the layer l , and $g'(z^{(l)})$ is the derivative of the activation function g . Therefore, $g'(z^{(l)}) = a^{(l)} \cdot * (1 - a^{(l)})$.

After finishing above process until the innermost layer we compute the average:

$$\begin{aligned} D_{ij}^{(l)} &:= \frac{1}{m} \delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ D_{ij}^{(l)} &:= \frac{1}{m} \delta_{ij}^{(l)} & \text{if } j = 0 \end{aligned} \quad (2.37)$$

Finally, we can show that each D is equal to the following

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}. \quad (2.38)$$

However, unlike the most in logistic regression reach the global minimum (convex), however, most situations in DNNs, the loss function is not convex shape; therefore, it can often end up in local minimum.

There are several algorithms to solve this issue called optimizer. As we mentioned above, the computation cost is too big to calculate all the loss. Therefore, we use a mini-batch concept in DNNs. Stochastic gradient descent (SGD) is the basic algorithm that checks gradient descent based on its batch size. However, SGD is sensitive to the learning rate, in other words, if the learning rate is low, it takes a long time to reach the minimum and if the rate is too high SGD cannot find the minimum.

Therefore, there are two types of approach to enhance SGD. First of all, the momentum approach [29] is using the momentum mechanism in physics. Thus, when the momentum based optimizer moves like SGD first, then it moves further to the direction that it moved before. The other approach is considering the step size. For example, when Adagrad [30] decides the next step, if the region in the loss function has been visited, it moves using small step, and if the location is newly visited, it moves using bigger steps than usual. In the end, Adam optimizer [31] is using the best of both world; in other words, it considers both the direction and the step size at the same time to decide the next step. In our project, we use Adam as our optimizer.

2.5.5 Activation Function

The key concept that drives the neural network is the activation function. As we mentioned above in many classification tasks, it is challenging to handle the non-linear data set. Therefore, the logistic regression is appeared to overcome this limitation of linear regression. And this non-linearity characteristics can be controlled by activation functions. In neural networks, each cell's output is wrapped by a specific activation function. There are several accessible activation functions such as sigmoid that we

mentioned earlier, hyperbolic tangent (tanh), and rectified linear unit (ReLU) and all functions show different shape as shown in 2.18.

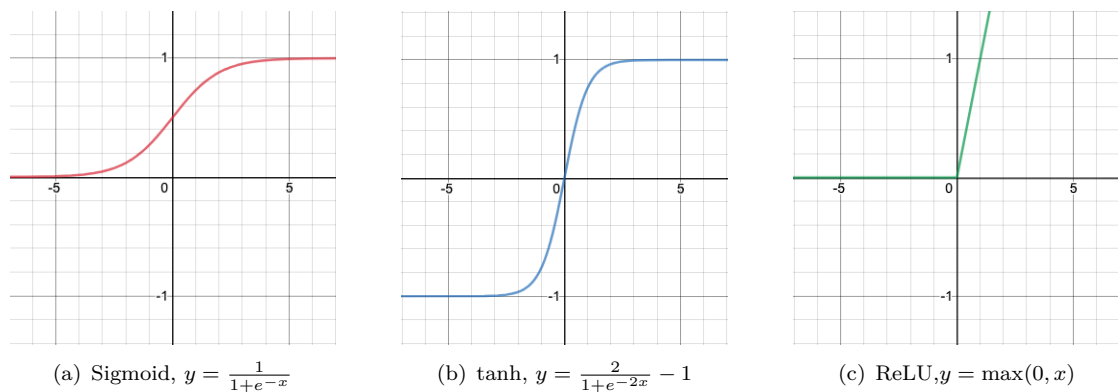


Figure 2.18: Different types of activation function

The tanh activation function is appeared to solve the limitation of the sigmoid activation function. Since all the output of sigmoid is a positive value $[0, 1]$, when it calculates the gradient descent, there are several directions that it cannot reach. This condition constraint the direction to find the minimum value; thus, it degrades the convergence speed. To overcome this issue, tanh moves its middle point to 0; therefore, the possible range is $[-1,1]$ now the moving direction of gradient descent is not constrained. However, both activation functions face the gradient vanishing problem. Because as shown in Fig.2.18(a) and Fig.2.18(b), if the input values are far right or far left in the graph, their gradient descent values are almost 0. Therefore, ReLU partially solves this issue using the shape in Fig.2.18(c); however, for the negative x it also faces the same problem. Leaky ReLU solves this issue by giving little gradient descent to the negative input parts.

2.5.6 Recurrent Neural Networks (RNNs)

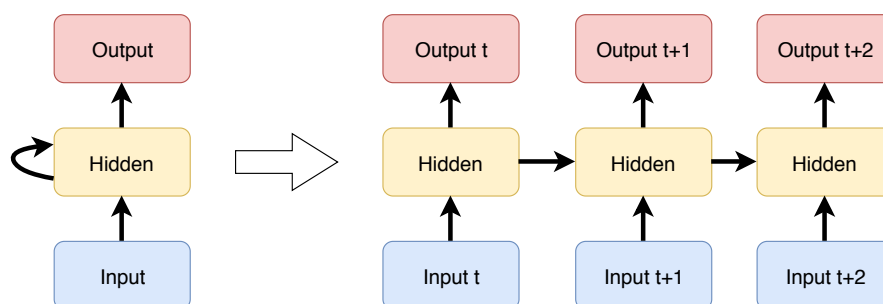


Figure 2.19: Recurrent neural networks

While FNNs show exceptional performance for the image recognition task, several challenges are difficult to be handled by FNNs. One case is when the data is highly related to the sequence or time series, e.g., natural language, speech, and values varying

by the time. Since we feed the entire data at once, the networks do not understand the sequence in the data. In order to do that, the input layer should change the number of nodes every time there is a new sequence to be trained. This is an impractical approach to scale up the model. To deal with this issue, the recurrent neural networks are used. The recurrent neural networks (RNN) are based on [29] and the underlying architecture is almost the same with the regular ANN except that the hidden layer takes the additional feature from the former state of itself. The networks can be visualized in two ways, as shown in Fig.2.19. The input will be sequentially fed into the networks ($t, t+1, t+2\dots$) and the corresponding outputs will be generated by taking an additional feature from the previous hidden node state.

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \quad (2.39)$$

$$y_t = \sigma_y(W_y h_t + b_y) \quad (2.40)$$

The equation of RNNs can be formulated as shown in 2.39. σ represents the activation function, and W denotes weights of each node while U means the vector representation from the input x . And b denotes bias like basic neural networks. The reason we use t instead of l in the equation is to highlight that the previous value is coming from the former state ($t - 1$), not of the prior layer ($l - 1$). In the end, the output y_t is calculated by multiplication of weight and hidden stated with additional bias.

There are several variations of RNNs, such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). The primary motivation of those two models is to resolve the vanishing and exploding gradient problem of regular RNNs [32]. Because the gradient quickly becomes zeros from all range since the repeated multiplication of the activation function in naive RNNs.

The inner architecture of LSTM cell is expressed in Fig.2.20. One significant difference between LSTM and regular RNN is the addition operation, unlike RNNs' multiplication operation. This element-wise addition will prevent vanishing gradient problem that multiplication occurs in regular RNNs. The LSTM can be formulated as shown in

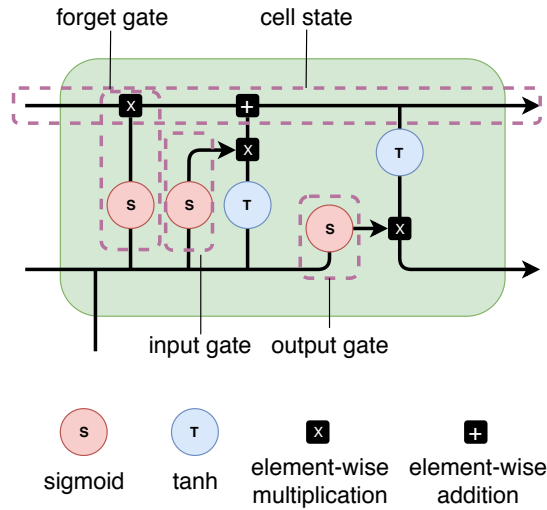


Figure 2.20: LSTM cell architecture

Eq.2.41.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \quad (2.41)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad (2.42)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (2.43)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \quad (2.44)$$

$$h_t = o_t \circ \sigma_h(c_t) \quad (2.45)$$

RNNs shows the excellent performance for the temporal data; in the meantime, there is a rise of a need for the neural networks, which has a better understanding in terms of spatial information.

2.5.7 Convolutional Neural Networks (CNNs)

FNNs and RNNs only consist of fully connected layers, and they often face the overfitting problem. Moreover, due to this ‘fully-connectedness’, they are easily fell into ‘curse-of-dimensionality’ problem. For example, if they use 1024 by 1024 pixels image with RGB, the parameters size easily reach 3 million. Also, FNNs cannot naturally consider the depth information (e.g., RGB color), either we use grayscale version, or we have to extend the input feature with each color manually. Either way, we will lose some essential local spatial information.

Furthermore, in the image recognition task, FNNs uses all the input features themselves. However, If the object is rotated, cropped, or scaled, it is challenging to recognize the target object. To overcome this problem, human has to generate extra dataset during the

pre-processing stage manually. Therefore, it is preferred to use the network model that can extract the information from the pattern rather than the pixel itself. This pattern extraction was often the task from human-side; however, CNNs automated this task.

The performance of CNNs is successfully proven in ‘ImageNet Classification with Deep Convolutional Neural Networks’ [33]. CNNs create a feature map that highlights the unique characteristics of the image. And the characteristic map is an input for the FNNs to classify the label that image belongs to. Therefore, the FNNs used in CNN is also called a classified neural network. To summarize, CNN can be said as a structure that the feature extraction neural network and the classified neural network are connected in series.

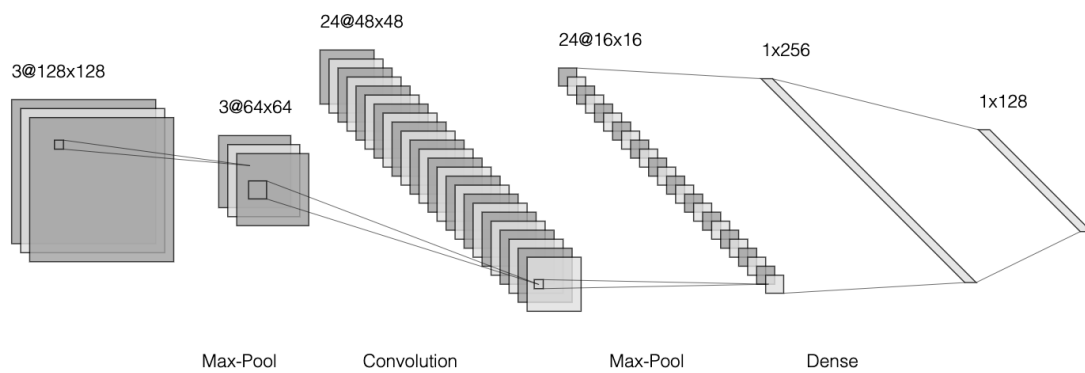


Figure 2.21: Example of CNNs architecture

Fig.2.21 shows the example of CNNs architecture, and there are several important concepts to understand the CNNs.

Convolutional is a mathematical combination of element-wise matrix multiplication and summation. If we formulate the process then the equation can be written as Eq.2.46 where K = Kernel, I = Image.

$$S(i, j) = \text{convolution}(I, K)(i, j) = \sum_m \sum_n K(i + m, j + n)I(m, n) \quad (2.46)$$

However, it is not intuitively clear how it is used in CNNs. Therefore, if we visualize the process, it can be shown as Fig.2.22.

The red box is the kernel layer, and the yellow box is the first input layer. The value inside the kernel layer is updated through the training process; we will describe that part more precisely later. While the kernel travels the image matrix to both axes, the element-wise multiplication will calculate the new value from the original matrix. Therefore, based on kernels value some values will be highlighted, and some values will be diminished,

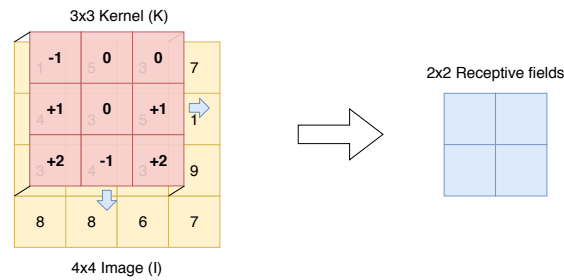


Figure 2.22: Convolutional process on 4x4 image using 3x3 kernel

and the decision is up to the target of the task. Finally, the summation of the newly calculated value will be assigned to the corresponding field in the next layer.

There are several configurations, also called hyperparameters during the convolutional process. First of all, as shown in Fig.2.22, after the convolutional process, the dimension decreases in the receptive fields from 4x4 to 2x2. However, sometimes, it is necessary to keep the original dimension; therefore, we use an additional operation called **padding**. the padding process is surrounding the original matrix with the zero-valued pad, as shown in Fig.2.23. The number of padding is adjustable on the intention of the convolution process; thus we can control the output volume.

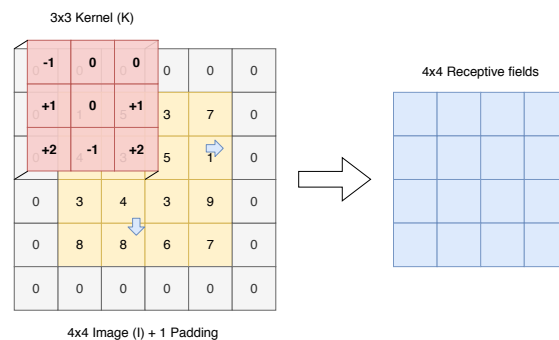


Figure 2.23: Padding

Secondly, we can also adjust the number of step for the convolutional layer while it is moving. The **stride** parameter decides how many pixels will be skipped for every traversal. The Fig.2.24 shows how 2-stride assigns the value from input fields to receptive fields while Fig.2.22 and Fig.2.23 show 1-stride movement.

The motivation of using stride higher than 1 is downsampling by reducing the overlapping pixels from the image or matrix [34].

Lastly, the convolutional process also contains the **depth** configuration also called **kernel size** for the output. This is one of the reasons that CNNs can learn more information from the less amount of dataset than regular FNNs. For example, if we set the output depth to 10 from 3 depth image (RGB), the output volume will be as Fig.2.25 (no

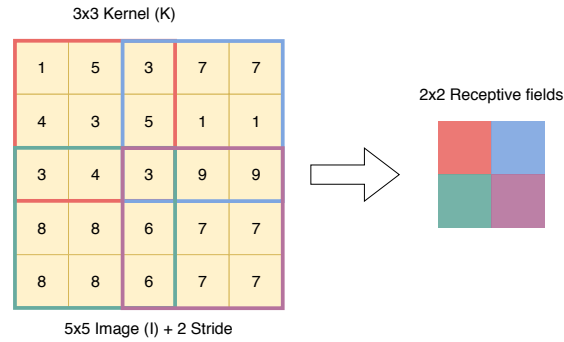


Figure 2.24: Stride

padding and one striding). The benefits of expanding the depth from the original input are learning a more various pattern from the previous layer. In other words, different neurons along the depth of output will be activated by different patterns such as edge, color, or orientation.

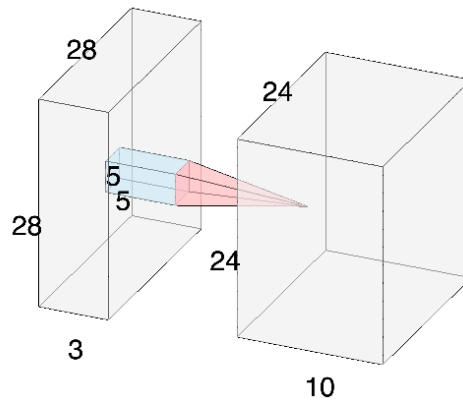


Figure 2.25: Depth

Therefore, we can calculate the output volume by considering these three hyperparameters: padding (P), striding (S), and depth (K) by following Eq.2.47.

$$\text{Output Volume} = K \frac{W - F + 2P}{S} + 1 \quad (2.47)$$

Additionally, dilation parameter is introduced in [35]. The motivation of dilation is since most CNNs had been originally developed to tackle the image classification task (only needs to differentiate). However, it is not the optimal model for the other task such as semantic segmentation which requires to understand the content in the image in the pixel level. To achieve the semantic segmentation, localization and detection should be followed after image classification to obtain additional spatial information such as the location of the object. In the end, the inferring the label for corresponding content in the image also called dense-prediction so that the model can segment each target

object. The reason previous CNNs' approach is considered inefficient for this task is, to gain the semantic information, it requires pixel-wise information from the bigger or global perspective. However, the only way to obtain that information is, rescaling and resampling the same pixels several times with different configuration. Therefore, instead of analyzing the multiple rescaled image, dilation concept is proposed.

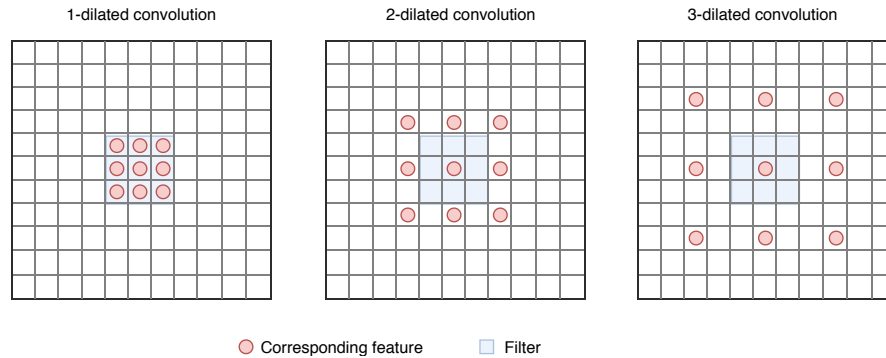


Figure 2.26: Dilation

The main idea of dilation is obtaining a bigger view of the filter while having the same size as the filter, as shown in Fig.2.26. When we use a 3x3 filter, and the red dots represent the corresponding pixel between the original image and filter. The left image shows our regular CNNs' behavior, and the middle image is 2-dilated CNN's behavior. As we can see, the corresponding pixels are 1 pixel further then 1-dilated CNNs and same as the third image (3-dilated CNNs). And this is why dilated CNNs offer a bigger view of the same size filter.

The dense prediction is highly related to our soft mask estimation in speech separation process [36]; therefore, in our project, we also use dilated CNNs to spectrogram.

Pooling is the process of reducing the dimension. When the features pass the pooling layer, it extracts the specific feature in the filter size. For example, if we use a 3x3 filter, nine feature will be selected from the previous layer, and they will be reduced into one feature based on pooling policy. The most common pooling policies are *Max pooling* and *Average pooling*. As we can assume based on the name itself, *Max pooling* select the highest value from those nine features and average pooling calculate the average of viewed features and pass the average value to the next layer as shown in Fig.2.27. Therefore, using pooling layer can reduce the number of parameters and computational cost than using the original dimension size.

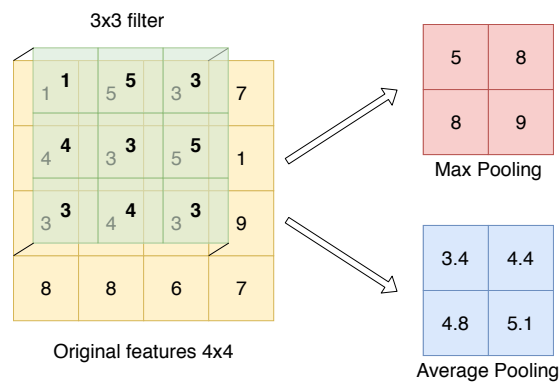


Figure 2.27: Example of pooling process

2.5.8 Embedding

Embedding is one of the successful feature derived from deep learning techniques. The primary motivation for using embedding is transforming the discrete value to continuous value. Unlike image recognition, which consists of a pixel-based dataset, many advanced techniques are challenging to be digested directly to DNNs models. For example, In natural language process (NLP), many researchers found the difficulty of representing the language data to the matrix or vector representation. The main obstacle is the curse-of-dimensionality. If we naively represent the one sentence as a vector such as "deep learning is fun", we can vectorize them into the vector shown in Fig.2.28. However, if we add the one more word to the sentence like "deep learning is very fun", we have to add one more row into the existing vector. More and more newly discover word will keep expanding the vector space.

$$\text{"deep learning is fun"} = \begin{bmatrix} \textit{deep} \\ \textit{learning} \\ \textit{is} \\ \textit{fun} \end{bmatrix} \tag{2.48}$$

$$\text{"deep learning is very fun"} = \begin{bmatrix} \textit{deep} \\ \textit{learning} \\ \textit{is} \\ \textit{very} \\ \textit{fun} \end{bmatrix} \tag{2.49}$$

Figure 2.28: Example of vectorizing the words

Furthermore, if we have a word "learning", and if we vectorize it using the same way with assigning binary value, the vector representation is , and it is called one-hot-encoding. One-hot encoding representation is traditionally used to handle the non-continuous variable. However, it has several limitations. First of all, it does not represent the

$$\text{“learning”} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad (2.50)$$

Figure 2.29: One-hot encoding

relationship between one word to the other word. If we have three words, cat, dog, and computer, the vector representation becomes

$$\begin{bmatrix} \text{cat} \\ \text{dog} \\ \text{science} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix}.$$

Intuitively, cat and dog should show more similarity than with science; however, one-hot encoding cannot represent this relationship. Secondly, the matrix or vector will be sparse, meaning that the matrix size should cover all the non-corresponding word to represent the single word, as shown in Fig.2.29. Therefore, the one-hot encoding approach can easily face the curse-of-dimensionality. In order to solve these limitations, we use dense representation (Fig.2.30), and this way of vectorizing the variable is called **embedding**.

$$\begin{bmatrix} \text{cat} \\ \text{dog} \\ \text{science} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 0.5 \\ 0.3 \\ -0.8 \end{bmatrix} & \begin{bmatrix} 0.2 \\ 0.7 \\ -0.7 \end{bmatrix} & \begin{bmatrix} -0.3 \\ -0.4 \\ 0.9 \end{bmatrix} \end{bmatrix},$$

Figure 2.30: Dense representation

There are three main benefits of using embedding. First of all, we can find the nearest neighbors in embedding space; therefore it can be used for association and clustering task. Secondly, the embedding itself can be used as an input of the machine learning task while avoiding the curse-of-dimensionality problem. Lastly, it is visualizable to express the relationship between the entities using a specific algorithm such as T-distributed stochastic neighbor embedding (t-SNE).

We have only described the word related example; however, this embedding concept is widely used in many other areas. For instance, in the speaker verification task, which requires to identify the corresponding speaker based on their speech, many research use embeddings called *i-vector* or *d-vector*. It is not simple to differentiate the speaker’s voice from the audio signal; therefore, the task also requires representative embedding space.

The embedding can be directly trained for a particular task (e.g., word2vec [37]). However, also generating embedding does not require any unique technique beyond the deep learning

process, because the embedding has been generated indirectly during the training process in any neural networks.

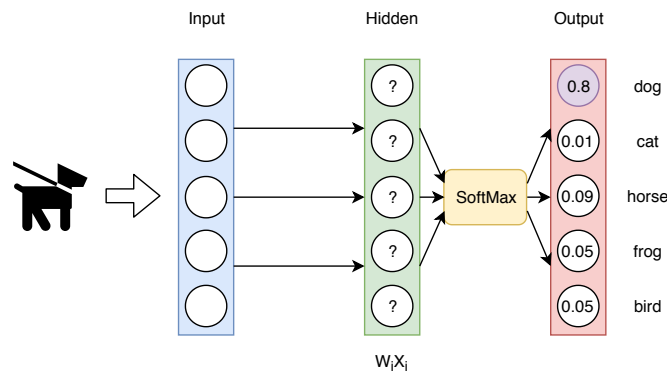


Figure 2.31: Simple neural network for image recognition task

In Fig.2.31, we introduce simple image recognition task. If the model is trained enough, the model will generate the different value right before it passed the softmax layer, which calculates the probability for the possible class. Then, if the weights in the hidden layer are trained enough, the multiplied result with input feature will indirectly represent the input itself. This group of weights in the final layer can also be seen as an embedding space since they can represent any types of input with a fixed dimension. The usage of embedding in our project will be introduced in the next section.

Chapter 3

Related Works

In this chapter, we introduce more related techniques to the speech separation in deep learning. First, we describe the overall learning process, and then we review three popular systems in speech separation task. In the end, we finish this chapter by introducing the three research that inspires our solution.

3.1 Conventional Techniques for Speech Separation

Before deep learning techniques start to be applied, there were several conventional approaches to tackle this problem. For instance, computational auditory scene analysis (CASA), the technique inspired by human's behavior to separate the speech [38], Non-negative matrix factorization (NMF) which is a data-driven approach using matrix decomposition [39], and Gaussian mixture model-hidden Markov model (GMM-HMM) [40] based generative models which have overcome the previous models' limitation such as temporal dynamics. Although, those techniques are not directly applied in modern technology, however, most of the foundations of current deep learning based approach are built from those conventional techniques.

3.2 Deep Learning in Speech Separation

3.2.1 Understanding the Task

First, if we denote the mixed signal, $y[t] = \sum_{s=1}^S X_s[t]$, where t is time index and $X_s[t]$ are each signal from different sources ($s \in \{1, 2, \dots, S\}$). Now the goal of the speech separation is separating individual $X_s[t]$ from mixed $y[t]$. As mentioned in Sec.2.3.2,

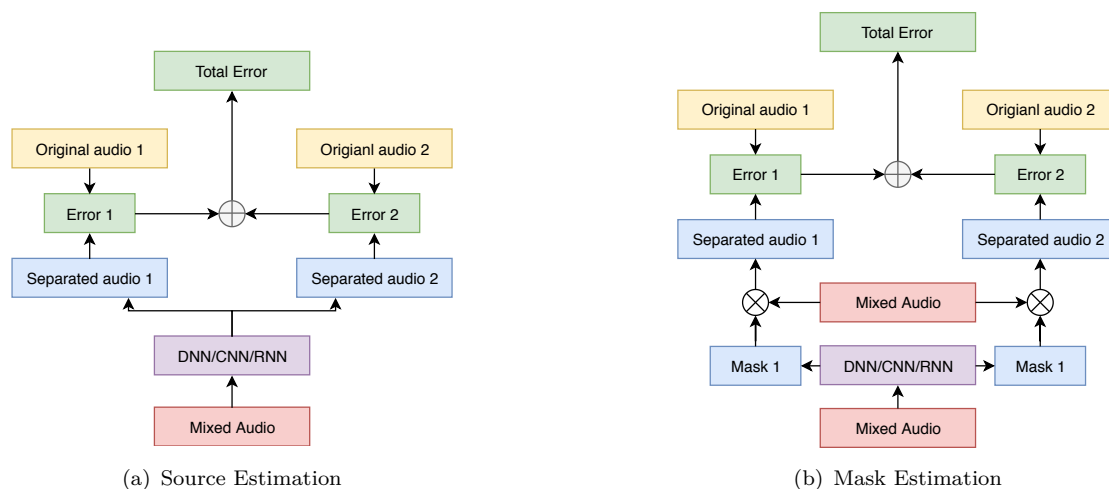


Figure 3.1: Source/mask estimation architecture to separate the original source

STFT is used as a main resources to represent audio data in many research, therefore, the task can be rewritten as recovering $X_s(t, f)$ from $Y_s(t, f)$ in the T-F domain for every time index t and frequency bin f .

However, since there is an infinite number of combination to obtain the same mixed signal, it is infeasible to separate the individual signal based on only combined signal information itself. To tackle this issue, the process should recognize the hidden pattern that can be seen a reasonable division of each source, therefore, this is why deep learning technology is considered as a core technique of the cocktail party problem.

Then, the task can be reformulated as a multiclass classification or regression. Even though the purpose of modeling is different, the reason that both can obtain the same task goal (separation) is, the task can be differentiated based on the soft mask that we are using as mentioned in Sec.2.3.3. If we plan to use IBM, the task can be considered as a binary classification model since all values will be either 0 or 1 for each time t and frequency f index. On the other hand, if we use different soft masks such as IRM, the task can be seen as a regression problem.

Even though we use soft mask estimation as our training target, however, it is also possible to train to estimate the individual source. However, many research has proven that the performance is better when the model estimates the soft mask instead of the target source itself [17, 19].

As shown in Fig.3.1, there are two approaches to apply DNN in source separation task. First approach is separating the target directly from the mixed audio (Fig.3.1(a)). We feed the mixed audio as an input of the network, and DNN estimates the target sources based on how much it trained. Then, the loss (e.g., MSE) will be calculated by comparing their original audio, which is part of audio that consists of mixed audio. In the end, the

final error will be summed by individual error, and the network will be trained to reduce the loss. The more similar estimation has a lower error.

The second approach is, as shown in Fig.3.1(b), the network can be trained to estimate the mask instead of the target source itself. Again, if we estimate the mask, we can reconstruct the source $\hat{X}(t, f) = \hat{M}(t, f) \otimes Y(t, f)$ where \otimes is the element-wise product and \hat{X}, \hat{M} denotes estimated version. The reason that masks estimation is preferred over the direct target estimation in the last years is, masks are more constrained and more robust to input variabilities caused by the non-normalized difference between signals such as energy differences than the target itself [6].

However, the mask estimation may have controversy since it is calculating between target audio and estimated audio not from between masks. For example, intuitively, it is reasonable to formulate the cost function like the following equation (F denotes Frobenius norm).

$$Cost = \frac{1}{T \cdot F \cdot S} \sum_{s=1}^S \left\| \hat{M}_s - M_s \right\|_F^2 \quad (3.1)$$

T = total time frames

F = total frequency frames

S = total number of sources

However, directly optimizing mask estimation has two problems [6]. First of all, the mask is difficult to be defined when both mixed and original source has a silence segment at the same time period (e.g., $|X(t, f)| = 0, |Y(t, f)| = 0$). Secondly, the error between masks does not guarantee the same amount of error between sources after its reconstruction. Therefore, many researchers [17, 19] proposed source based loss functions as shown in Eq.3.2.

$$Cost = \frac{1}{T \cdot F \cdot S} \sum_{s=1}^S \left\| \hat{X}_s - X_s \right\|_F^2 \quad (3.2)$$

$$= \frac{1}{T \cdot F \cdot S} \sum_{s=1}^S \left\| \hat{M}_s \otimes Y_s - X_s \right\|_F^2 \quad (3.3)$$

If we summarize this basic concept of speech separation using DNN, there are two main points. Firstly, DNN will be trained to generate a soft mask that can reconstruct the target source from the mixed source. Secondly, the loss or cost function will be derived

by the target and estimated source, not masks themselves. The variety of techniques can be obtained by additional input, augmented feature, or network architectures. However, the fundamental of most DNN based speech separation tasks is the same as above.

3.2.2 Label Permutation Problem

The model that we are proposing later in this paper has a reference that the model can target the speech. In other words, we offer an explicit reference that the model can identify the target that should be separated from the mixed source. However, this condition is not offered in most cocktail party problems.

The label permutation problem is a well-known issue in blind source separation task. Assume that the mixed audio contains two speakers, X_1 and X_2 . Since we have no identifiable information of each speaker during the training process it is not possible to assign correct target for each speaker. In other words, \hat{X}_1 can be compared with X_2 and vice versa. If there are more speakers than two, the combination grows quadratically.

Some past research [41] shows that DNN can learn its way to handle this ambiguous problem eventually. However, it is inevitable to offer the advanced front-end to the model to match the right target during training to improve both performance and training speed.

From the next section, we review three techniques in speech separation that solves the label permutation problem.

3.2.3 Deep Clustering

Deep clustering (DPCL) is proposed by [13] in 2016, and the first well-known techniques to address the permutation invariant problem using deep learning. The main idea of DPCL is instead of considering the task as class-based learning; they approach it as partition-based learning. To understand DPCL clearly, it is essential to understand the motivation that they approach the problem into partition-based over classed based. The class-based learning is training from the known labeled object while partition-based is learning label without having the labeled object. It can be also seen as the difference between supervised learning and unsupervised learning. However, the speech separation is still supervised learning task since it has the clean audio that can aim to train.

The benefit of partition-based learning is that it can segment unknown objects. Even though the class-based approach is straight forward and has been successful in my different area, however, in a real-world situation, it is often difficult to know the precisely the label

or it has a large number of possible labels. For example, signal data is not possible to have an explicit representation. Therefore, to simplify the learning process, DPCL uses deep learning to derive embedding features instead of using specially designed features that can represent the speaker. This embedding space is not intuitively explainable; however, as covered in Sec.2.5.8, it vectorizes the features to represent unstructured data.

DPCL assumes that each T-F bin of the mixed speech belongs to only one speaker. Therefore, the mixture spectrogram is segmented into clusters for each speaker. If we define the given mixture audio as $Y_i = g_i(y)$, ($i \in \{1, 2, \dots, N\}$), where i is the T-F bin (t,f), DNNs are trained to generate the D-dimensional embeddings (V) from input signal (Y), $V = f_\theta(Y) \in \mathbb{R}^{N \cdot D}$, where each row vector $|v_i| = 1$ has unit norm. Now the embeddings V can be used to estimate N by N affinity matrix VV^T . Then we generate the target affinity matrix EE^T , which is created by using labeled indicator such as IBM from Sec.2.3.3. Now the loss function can be written as:

$$\text{Loss} = \|VV^T - EE^T\|_F^2 \quad (3.4)$$

$$= \|V^T V\|_F^2 - 2\|V^T E\|_F^2 + \|E^T E\|_F^2 \quad (3.5)$$

where $\|\cdot\|_F^2$ denotes the squared Frobenius norm. DNNs will be trained to reduce the difference between VV^T and EE^T .

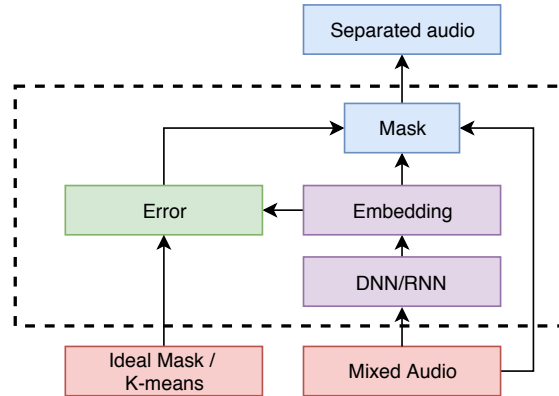


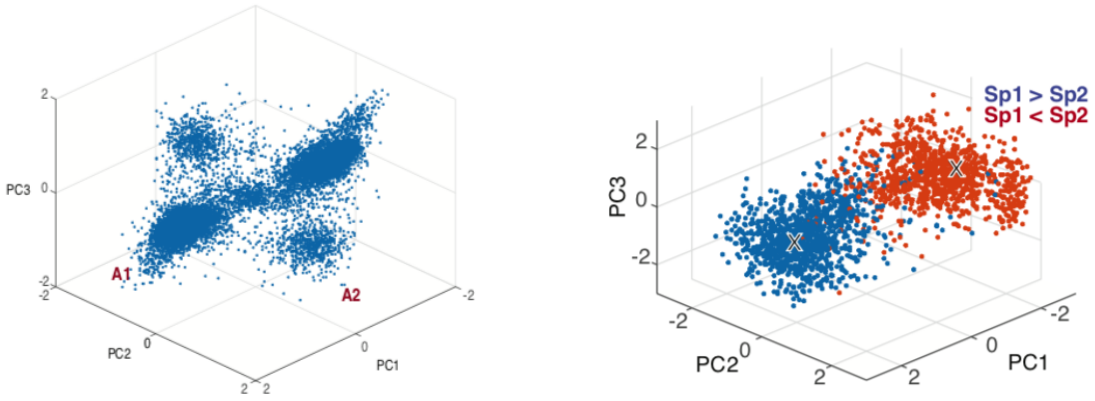
Figure 3.2: DPCL system architecture.

In the inference stage, first, we generate the embeddings from mixture Y and cluster them into individual sources using K-means. Assigning ‘0’ and ‘1’ to the corresponding spectral bins of each cluster will form the ideal binary mask. Then element-wise multiplication between estimated binary mask and mixture will extract the individual audio. The entire process is shown in Fig.3.2.

3.2.4 Deep Attractor Network

Deep Attractor Network (DANet) is introduced to improve several limitations and inefficiency of DPCL [42]. One of the main drawbacks of DPCL is the absence of end-to-end mapping. In other words, DPCL is optimizing the affinity matrix of embeddings, not the separated audio itself.

DANet is inspired by the perceptual magnet effect [43], which is the human perceptual ability that can warp the stimulus space to pull the closest sounds. Therefore, they use the concept called **attractor** in their architecture. The principle of the attractor is setting the reference point for each source in the embedding space which pulls all the T-F bins to itself. Then, the similarity between embedded point and each attractor (per source) is used to estimate the mask for each source.



(a) Location of attractor points in the embedding space using the first three principal components

(b) Location of T-F bins in the embedding space using the first three principal components where colors distinguish the power of speakers

Figure 3.3: PCA result of speech embedding space, figure from [42]

It is more evident if we check Fig.3.3. Each dot represents each audio, and they are located using the first three principal components from the embedding vector generated from the neural networks, in this case, LSTM. In Fig.3.3(a), we can observe two distinct attractor pairs A1 and A2. If we visualize the T-F bins with two speakers in the embedding space, we can also observe the two clusters, and two attractor points are marked as X.

The attractors $A \in \mathbb{R}^{S \cdot K}$ can be formulated with embedding space $V \in \mathbb{R}^{F \cdot T \cdot K}$ and dominant source membership $E \in \mathbb{R}^{F \cdot T \cdot S}$ as following:

$$A_{s,k} = \frac{\sum_{t,f} V_{k,ft} \cdot E_{s,ft}}{\sum_{t,f} E_{s,ft}} \quad (3.6)$$

where s, f, t and k denote the number of speakers, frequency, time and embedding dimension respectively.

Then, the mask can be estimated by softmax of the embedding space and attractors as

$$M_{f,t,s} = \text{Softmax} \left(\sum_K A_{s,k} \cdot V_{f,t,k} \right). \quad (3.7)$$

Now if we formulate the loss function using the estimated mask, the loss function can be written as

$$L = \sum_{f,t,s} \|X_{f,t,s} - Y_{f,t} \cdot M_{f,t,s}\|_2^2. \quad (3.8)$$

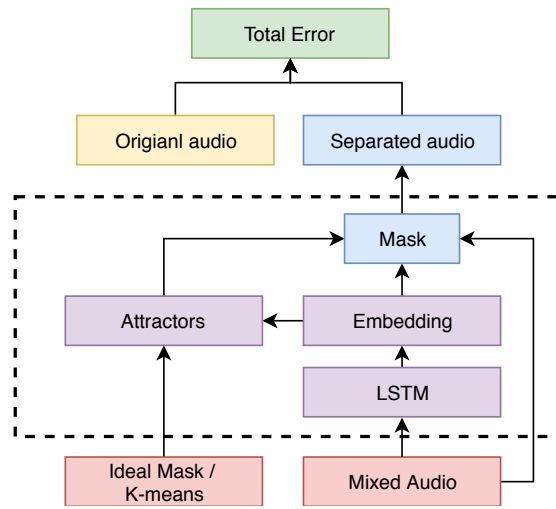


Figure 3.4: DANet System Architecture

Unlike the DPCL, as shown in Fig.3.4, DANet calculates the loss between the separated source and the target source. However, DANet is still the extended version of DPCL; therefore, it cannot solve the limitation that it needs to know the number of speaker in the mixture.

3.2.5 Permutation Invariant Training (PIT)

PIT is introduced to tackle the label permutation problem differently, and it can directly minimize the separation error [24]. The main idea of PIT is shown in the dashed box in Fig.3.5. There are two output stages. First of all, as we demonstrate in Fig.3.1(b), estimate the individual mask for each speech and reconstruct the clean speech (output 1). Then, the pairwise MSE is calculated using ground-truth audio and reconstructed audio for any possible assignment (e.g., $X_1 = \hat{X}_1$, $X_2 = \hat{X}_2$ or $X_1 = \hat{X}_2$, $X_2 = \hat{X}_1$). Next, the model chooses the pair that shows the least MSE error, which can be seen as the right match. After deciding the least MSE pair, the model is trained for the further reducing

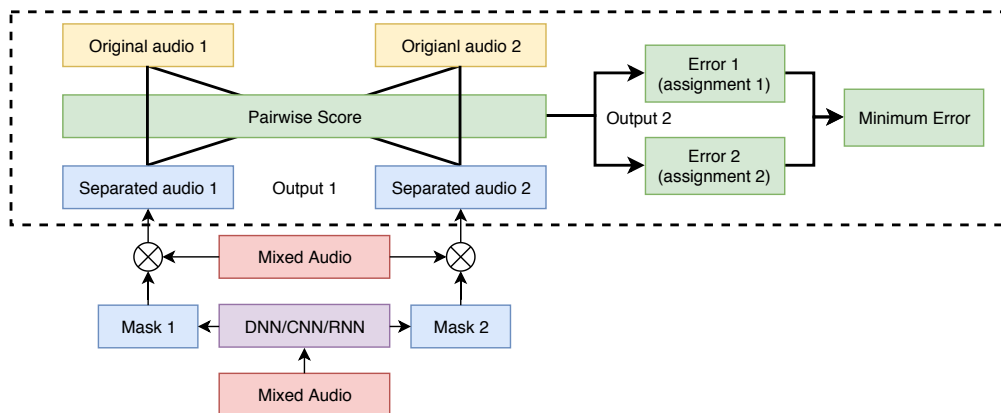


Figure 3.5: PIT System Architecture

of the assigned loss function shown in Eq.3.9:

$$MSE = \frac{1}{F \cdot T \cdot S} \min_{s' \in \text{permu}(S)} \sum_{s=1}^S \left\| \left| \hat{X}_s \right| - \left| X_{s'} \right| \right\|_2^2 \quad (3.9)$$

where $\text{permu}(S)$ is the permutation of speakers 1 to S .

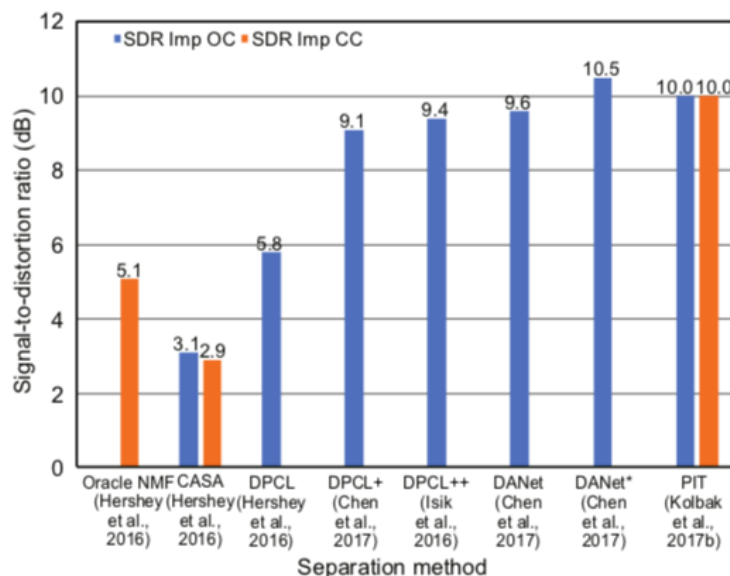


Figure 3.6: SDR improvements for different speech separation system using WSJ0-2MIX dataset, the figure from [6]

Fig.3.6 shows the performance comparison of speech separation models using the same dataset (WSJ0-2MIX). Unlike the other two systems (DPCL and DANet), PIT can perform well even in the closed condition, which is not knowing the prior knowledge about the speaker (e.g., number of speakers). Therefore, although DANet+ shows slightly better performance (10.5) than PIT (10.0), PIT can be more practical in a real-world situation. Moreover, the simplicity of implementation can help to integrate the other

method. Therefore, most recent works are based on PIT’s architecture with minor modifications [3, 44–46].

We have reviewed three popular models from their architecture and the approach to solve the label permutation problem. Even though we express as the PIT is the better models than others, all the models are still used as a foundation of much ongoing research [47].

3.3 Inspirational Works and Theory

In this section, we introduce several studies that can theoretically support our suggestion. First of all, we review the joint training system, which consists of speech separation and speech recognition. Secondly, to support our dynamic weighting loss function approach, we review the reweighting loss function related works, thirdly, we review the speech recognition system that uses WER into their loss function.

3.3.1 Joint Training with speech recognition system

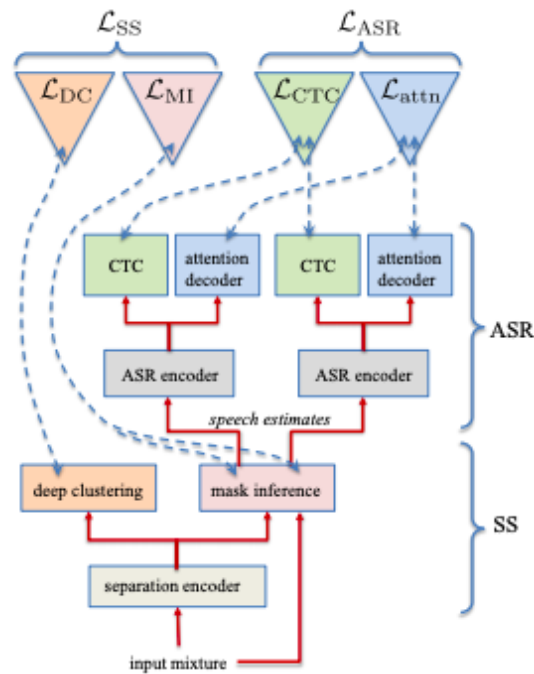


Figure 3.7: Joint training architecture using both speech separation and speech recognition, the figure from [48]

First of all, many researchers have suggested that integrating speech separation system with speech recognition can achieve further improvement [3]. Because the automatic speech recognition (ASR) requires speech separation system to perform correctly in a real-world environment, in the same manner, the speech separation system also requires

ASR to confirm its separated output with the transcription-level. Although these two systems can be trained separately and used after that, it is a reasonable idea to train both systems jointly.

The first fully jointed training technique is introduced in [48]. The most significant point of this architecture 3.7, it manages to build the end-to-end training process even with the integration of many systems. For the speech separation part, they use DPCL with chimera++ [49], and for the speech recognition part, hybrid of attention and CTC model introduced in [50] is used.

To integrate two systems, moreover, each system is also a combination of different techniques, it is important to combine the loss functions properly. The loss functions are shown from Eq.3.10 to Eq.3.15

First, we formulate the loss function for DPCL and Chimera++ network. Unlike the loss function that is originally proposed in Sec.3.2.3, they use the improved version [51] as shown in Eq.3.10.

$$L_{DC,W}(V, Y) = D - \text{tr}((V^T V)^{-1} V^T Y (Y^T Y)^{-1} Y^T V) \quad (3.10)$$

$$L_{MI,tPSA,L_1} = \min_{\pi \in P} \sum_a \left\| \hat{M}_c \otimes |Y| - T_0^{|Y|} (|S_{\pi(c)}| \otimes \cos(\theta_Y - \theta_{\pi(c)})) \right\|_1 \quad (3.11)$$

Now, if we combine two loss functions above it can be written as:

$$L_{ss} = \alpha_{DC} L_{DC,W}(V, Y) + (1 - \alpha_{DC}) L_{MI,tPSA,L_1}, \quad (3.12)$$

where α_{DC} is the hyperparameter to adjust the relative importance of each loss.

$$p_{ctc}(C|X) = \sum_Z \prod_t p(z_t|z_{t-1}, C) p(z_t|X) p(C) \quad (3.13)$$

$$p_{att}(C|X) = \prod_l p(c_l|c_1, \dots, c_{l-1}, X) \quad (3.14)$$

For the speech recognition systems, the loss function can be written as in 3.15. p_{ctc} is likelihood for CTC based speech recognition system, likewise p_{att} is for attention based model. Same as Eq.3.12, the combined loss function can be written as:

$$L_{ASR} = -(\lambda \log p_{ctc}(C|X) + (1 - \lambda) \log p_{att}(C|X)), \quad (3.15)$$

where λ is the hyperparameter to regulate the importance of each log-likelihood.

Therefore, the final loss function will be

$$L_{SS+ASR} = L_{SS} + \alpha_{ASR}L_{ASR} \quad (3.16)$$

where α_{ASR} is hyperparameter to adjust the importance of L_{ASR} .

It is remarkable work that they manage to combine two systems in end-to-end architecture. However, since there are four different types of loss function, the training process is challenging to be analyzed to improve the specific part further. Even though the end-to-end model is efficient to train, the difficulty of implementation is the drawback.

3.3.2 Minimum Word Error Rate Training

Lastly, as we point out several times, in ASR area, there is a gap between the performance measurement and loss function. This problem is also pointed out in the speech recognition system [52]. Most sequence-to-sequence (e.g., attention-based) model based speech recognition systems use cross-entropy as their training criterion. However, the final measurement is WER or CER. Therefore, they propose a new training method to narrow this mismatch.

In the speech recognition task, we denote the set of input features in utterances as: $x = (x_1, x_2, \dots, x_T)$, where $x_i \in \mathbb{R}^d$. The corresponding ground-truth is denoted as: $y^* = (y_0^*, y_1^*, y_2^*, \dots, y_{L+1}^*)$, where $y_i^* \in G$ (G is the graphemes: the smallest unit of words [53]). Then we assume G contains the two special labels $\langle sos \rangle$ and $\langle eos \rangle$ mean the start of sentence and the end of sentence respectively. Now, the common loss function for attention based speech recognition system is

$$L_{CE} = \sum_{(x, y^*)} \sum_{u=1}^{L+1} -\log P(y_u^* | y_{u-1}^*, \dots, y_0^* = \langle sos \rangle, x). \quad (3.17)$$

The training criterion is maximizing the log-likelihood for the training dataset. Therefore, to directly minimize the WER, they propose a modified loss function using WER. First, Eq.3.18, shows the *expected number of word errors* over training dataset.

$$L_{werr}(x, y^*) = \mathbb{E}[W(y, y^*)] = \sum_y P(y|x)W(y, y^*) \quad (3.18)$$

where $W(y, y^*)$ denotes the number of word errors in the hypothesis. However, since there will be too much computational cost to calculate the possible word errors for the entire datasets, they use two approximation method called sampling and N-best. As their name suggests, sub-sampling the potential number of word errors from different

approaches. However, since we do not use the sampling method, we do not dive into the detailed methodology here. Finally, the joint loss function that they suggested is

$$L = \sum_{x,y^*} L_{werr}^{sample}(x, y^*) + \lambda L_{CE} \quad (3.19)$$

where λ is the hyperparameter to adjust the importance of cross-entropy loss. With several experiments, this new loss function can improve 8.2% WER.

Therefore, we expect that a similar approach will improve the performance of the speech separation task as well.

3.3.3 Reweighting the loss function

Finally, as we have mentioned several times, our solution is updating weights of loss function dynamically using WER. However, most research use weighted loss function, do not change the weight dynamically since the primary purpose of using weighted loss function is to handle the imbalanced dataset. They analyze the distribution of dataset and fix the importance of each class in the initial stage. Therefore we investigate the related works that prove that dynamical reweighting can be applicable in DNNs.

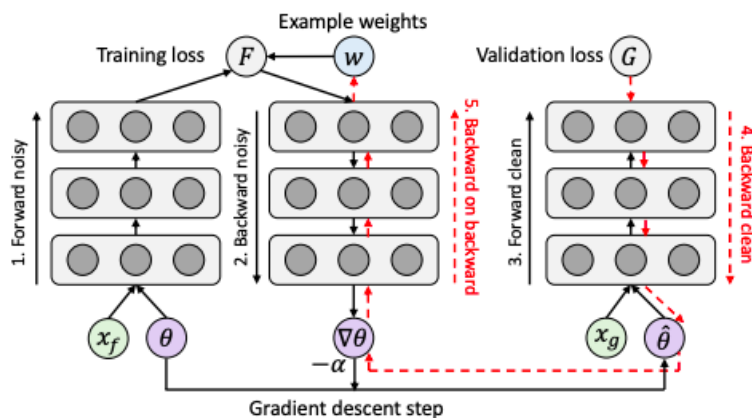


Figure 3.8: Diagram of reweighting algorithm, figure from [54]

In [54], they introduce the learning algorithm that can assign weights to training datasets based on their gradient directions. As shown in Fig. 3.8, the basic training process is the same as regular DNNs. However, the main difference is that the loss of the validation process is involved in the training process. Therefore, the model is outperforming compare to the regular algorithm when the training dataset is imbalanced, or the labeling is corrupted. The detailed mathematical proof is not covered in this paper; however, the main idea is they calculate the similarity between input data and validation data, and if they show the same gradient descent direction they up-weight the corresponding label,

otherwise they down-weight. Therefore, the gradient descent is following the route the shows the consistent performance for both training and validation dataset.

We can reframe the above approach to ours. When both SDR and WER show better performance, we reduce the weight of corresponding audio, and the other case, we increase the weight. Therefore, the training process can perform more sensitively to the (performance-wise) mismatched dataset. The comprehensive approach is described in the next chapter.

Chapter 4

Solution Approach

4.1 Introduction

Before we analyze the existing approach and propose our solution, it is necessary to clarify our task again, because speech separation task has various constraints and assumption such as speaker dependency, the number of speakers, length of utterance and etc.

First of all, the model is under the speaker dependent situation; in other words, we have the necessary information that the model can identify the target to segregate. The details about the reference will be described more precisely in the following subsection.

Secondly, the number of speakers is constrained to two. Even though final inference can perform with more than two simultaneous speakers, to compare with other research under the same condition, we limit the number of speakers intentionally.

Thirdly, the model separates only the targeted speech, not the other noisy speech. Therefore, this task can also be called speech enhancement.

Lastly, the final architecture is not an end-to-end model, which means that the training process is not starting and finishing internally. The model is combined with other systems which trained separately.

4.2 Existing Approaches/Baselines

4.2.1 Ideal Ratio Mask

As mentioned in Sec.2.3.3, IRM is the best case that speech separation system can generate through training. However, it is not realistic to create the same soft mask as

IRM through the model. We use IRM for reconstructing the target audio and evaluate the performance. Therefore this performance will be the oracle system of our model.

4.2.2 VoiceFilter

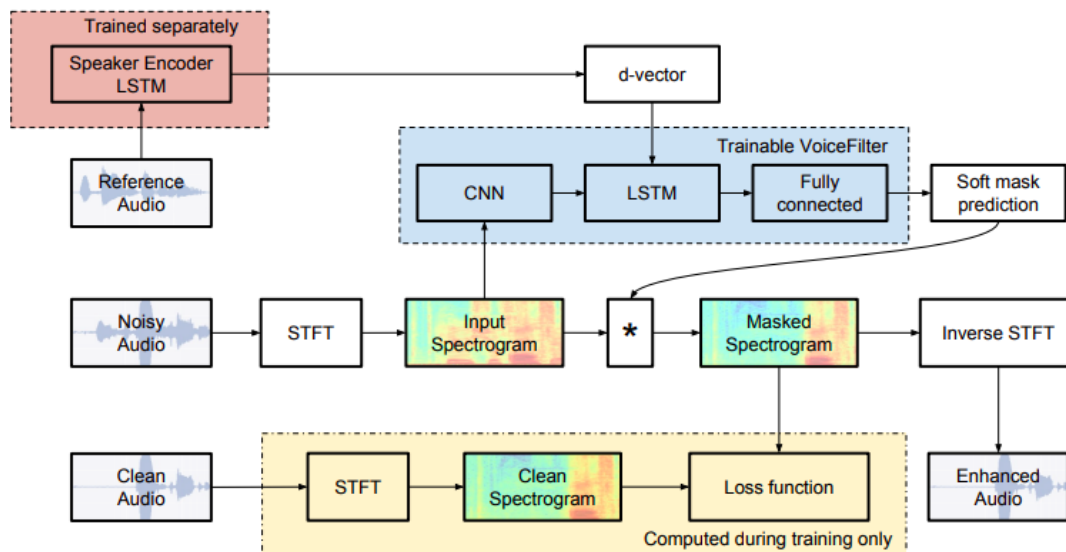


Figure 4.1: VoiceFilter System Architecture from [3]

Voicefilter is proposed in 2019 from Google [3]. This system is speaker conditioned separation. In other words, the system requires additional information related to the target speaker that we want to separate from the mixed audio. In the paper, they use *d-vector*, which is introduced in the speaker verification task [55].

The entire architecture of Voicefilter is shown in Fig.4.1. The model requires three audio files. First, the **clean audio** is the target or can be called the ground truth that should be separated from the noisy audio. The audio file will be transformed into spectrogram using STFT.

Second, the input data for the networks is **noisy audio**. The way to generate this noisy audio will be described more precisely later in Sec.5.1.3. Same as the clean audio, the spectrogram will be generated first. Then the spectrogram is fed into networks to train the model.

The last audio file is **reference audio**. This audio file is used in a separated system, which called speaker encoder. The purpose of this LSTM (red box in Fig.4.1, is to generate the embedding called *d-vector*.

The idea of *d-vector* is finding the vectorized representation of a certain speaker. It is originally used for the speaker verification task (*Who is speaking?*). The speaker

verification task is a classification task in machine learning [56–58], therefore, there can be many types of network architectures such as FNNs, CNNs, and RNNs. The d -vector is the average of the activation vectors in the last hidden layer as shown in Fig.4.2.

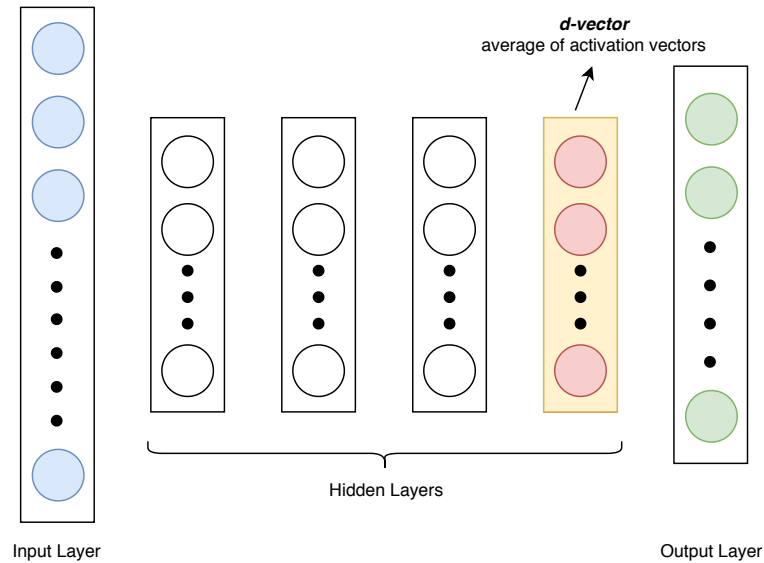


Figure 4.2: The location of d -vector in the neural networks

This speaker representation can also be seen as a voice print [55]. Once the model is trained to reach enough equal error rate (EER), in the paper 3.55%, the model can be used to generate the d -vector of the target speaker using the reference audio.

In the actual training process (blue box in Fig.4.1), first the network convolute the spectrogram to increase the volume of the original feature and the final layer pass the data into LSTM. Now the LSTM will be trained using both d -vector and convoluted spectrogram. This is the main difference between regular PIT and Voicefilter. As we mentioned above in Sec.3.2.5, PIT decides the target using K-means clustering, however, the speaker is targeted by this d -vector in Voicefilter. Therefore, Voicefilter does not face the label permutation problem that mentioned in Sec.3.2.2.

Finally, the soft mask will be generated in the last two fully connected layers, and the loss between target spectrogram and estimated spectrogram is calculated. Again, the loss, such as MSE and L2-loss, is not calculated by the difference between the estimated mask and IRM. It is calculated by the difference between target spectrogram and reconstructed spectrogram. Note that the reconstructed spectrogram is the result of element-wise multiplication of noisy spectrogram and estimated mask.

The separation networks are based on [59], and it consists of eight convolutional layers, one LSTM/BLSTM layer and two fully connected layers. The detailed information is in Table.4.1.

Layer	Padding		Kernel		Dilation		Filters / Nodes
	time	freq	time	freq	time	freq	
CNN1	0	3	1	7	1	1	64
CNN2	3	0	7	1	1	1	64
CNN3	2	2	5	5	1	1	64
CNN4	4	2	5	5	2	1	64
CNN5	8	2	5	5	4	1	64
CNN6	16	2	5	5	8	1	64
CNN7	32	2	5	5	16	1	64
CNN8	-	-	1	1	1	1	8
LSTM	-	-	-	-	-	-	400
FC1	-	-	-	-	-	-	600
FC2	-	-	-	-	-	-	600

Table 4.1: Parameters of the VoiceFilter, table from [3]

In the original paper [59] that VoiceFilter refers to the architecture, the optimal network configuration is decided by Vizier parameter optimizer [60], which is attempting many possible combinations of hyperparameters and recommend the best performing combination. Since the configuration is an experimental result, therefore it is not clear to define the role of each layer, (like other complex architectures), however, we can explain the motivation of the design from the higher level of view. First, CNNs expands the dimension from the original spectrogram so that we can obtain more information. The interesting fact from CNN1 and CNN2 is, each kernel focuses more on time and frequency domain, respectively, as shown in Fig.4.3. From CNN3 to CNN7, it is **dilated CNNs**

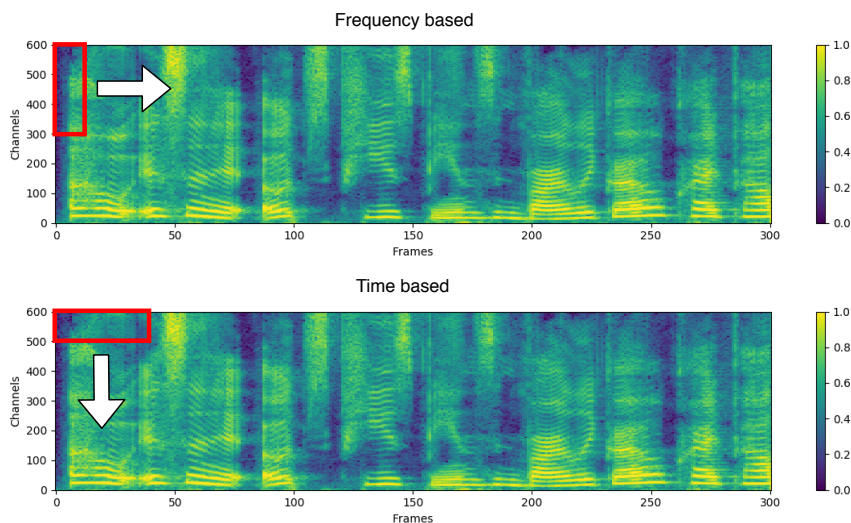


Figure 4.3: Time and frequency focused kernel size

for time domain that we have described in Sec.2.5.7. Since in speech data, the time domain is relatively more important than other stationary signals, therefore to obtain a bigger view of time-domain the dilation rate keeps increasing through the deeper layers.

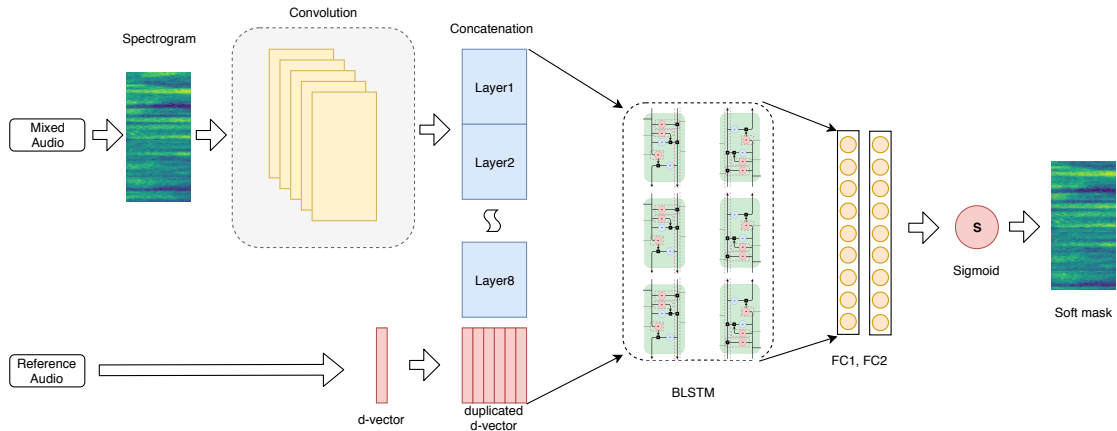


Figure 4.4: Layer-level training process

The layer level training process is shown in Fig.4.4. The dimension of the last layer is $[8, \text{time}, \text{freq}]$, the batch size is one, and 8 is the depth of the last layer. Then, we change the dimension to $[\text{time}, 8 * \text{freq}]$ before we pass the value to the LSTM layer. Later then, we concatenate the result of CNNs and *d-vector* from the reference audio. Finally, this concatenated tensor will be fed into LSTM. Since we use Bidirectional LSTM, the result dimension is $[\text{time}, 2 * \text{lstm-dimension}]$. After pass through two fully connected layers, we estimate the softmask using the sigmoid function for each corresponding pixel ($[\text{time}, \text{freq}]$). The label permutation problem is solved in this LSTM layer when the speech is trained together with reference *d-vector*.

4.3 Proposed Solution

In this section, finally, we introduce our novel approach. There are two significant contributions, first of all, we propose the loss function that can consider both transcription-level and signal-level performance. Secondly, we suggest our generalized integration architecture that applies to any other systems.

4.3.1 Performance Based Weighted Loss Function

As we have mainly highlighted loss functions in Sec.2.5.2, and Chapter 2.2, our core solution is also based on the modified loss function that can dynamically reflect the transcription-level performance.

There are several studies that we have mentioned in Sec.3.3, and our solution is inspired by them. The primary motivation is, there is a gap between the loss function and the final performance measurement in ASR research as many researchers have pointed out.

Again, WER is the performance measurement; however, the loss function is based on log-likelihood. Even though there is a high correlation between those two measurements, it is still sub-optimal. Likewise, the major performance measurements of the speech separation task are SDR, SIR, and STOI, however, many models still use regular MSE or L2-loss function.

Therefore, we propose the new loss function,

$$WMSE = \frac{1}{n} \sum_{i=1}^n (\alpha + \text{WER}_i)(Y_i - \hat{Y}_i)^2 \quad (4.1)$$

where α is the hyperparameter, and we choose 0.3 based on our experiment.

The primary motivation of the proposed loss function is that the model can directly consider both signal-level and transcription-level performance. To do that, we use the weighted loss function mechanism, which is mostly used to tackle the imbalance dataset problem [61–64]. If we have a smaller dataset for the particular class, by multiplying higher weight to that class than other classes, we can adjust the loss function to react more strongly for the lacking class. In our case, the weight is calculated using WER; therefore, speech that has higher transcription-level error will be penalized more than its original signal-level difference. In other words, our loss function is raising the correlation between loss and WER more than existing solution VoiceFilter.

The hyperparameter α is used based on our two assumptions. First, we add α to differentiate the performance between the original model and our proposed from the early stage, because when training starts, in the early stage, most separated speech's WER is 1.0 which means it will follow the original model's learning process. Therefore, to penalize additionally, from the beginning of the training process, we add extra value. Secondly, when WER becomes lower, this additional penalty will highlight the difference between well-separated speech and poorly-separated speech. We assume the convergence will occur later than the original model. In other words, our proposed model will converge at a lower point than the existing solution.

4.3.2 Generalized Pipelining with Speech Recognition System

The end-to-end training system has the benefit in terms of simplicity of system architecture and training process. However, it is difficult to integrate with the external system because every component should be implemented into the internal system architecture [65]. Even though there are many open sourced project with state-of-the-art machine learning techniques, most of the original authors do not open their source code for some reasons. Therefore, either we have to implement by ourselves or use the unofficial private source.

Even if we find the applicable code, it does not guarantee that they are implemented using the same framework since there are several deep learning frameworks such as Pytorch and Tensorflow. In this section, we introduce the architecture that is easily scalable and integrable with any other project.

In our project, the main challenge in implementation is the integration between existing speech separation (VoiceFilter) and speech recognition system (Deep speech) while minimizing the training performance degradation. The inspiration has come from Google speech-to-text API, which is RESTful API [66] and can be easily used, regardless of the system environment. Therefore, we load our speech recognition model on top of the API server implemented by the Flask framework as shown in Fig.4.5. The first

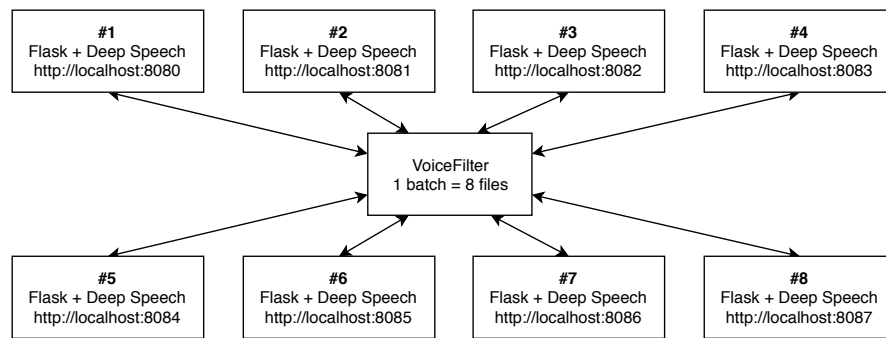


Figure 4.5: Simultaneous request for speech recognition

reason that it is beneficial to load the model and to build the RESTful API is, we do not have to consider the time until the model is loaded. For example, naively implemented inference code is mainly for one sample test data; therefore, if we want to inference multiple test dataset, we have to load the trained model every time. Sometimes it takes a longer time than the actual inferencing process. Note that the inference process can be continued without reloading the model if we implement the inference code the way that training process works such as using batch concept. However, the inference process is still sequential. Typically, while we are training the model, one batch contains several datasets, if we naively generate the transcription, the training time will be increased linearly, and this is constraining the batch training concept. In other words, we cannot increase the batch size. Moreover, compared to the GPU process, calculating WER can consume abnormally longer time than its core training time. Therefore we solve this issue with running multiple RESTful API servers that can generate the transcription. From the core training process's point of view, it can simultaneously request the WER value. Therefore, the batch size can be increased as much as GPU memory can load them since the time to calculate the WER for one file is the same as the multiple files.

Furthermore, the strength of our architecture is it can become exchangeable with other systems quickly without considering the system environment. Therefore, the training

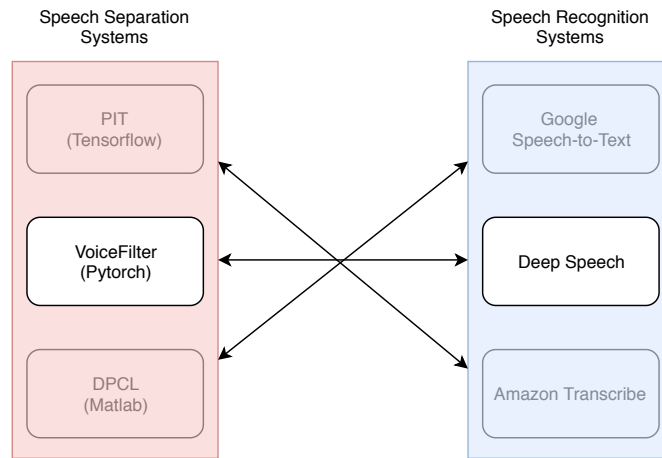


Figure 4.6: Exchangeable architecture with other systems

approach can be applied to any other speech separation systems (e.g., DPCL and PIT) likewise, WER scores can be calculated using other promising systems such as Google speech-to-text and Amazon Transcribe.

4.3.3 System Architecture

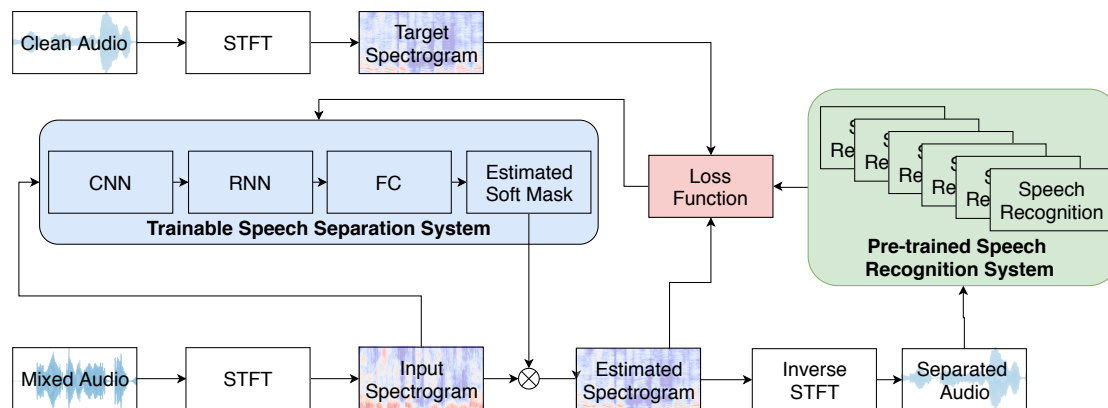


Figure 4.7: System Architecture

Our solution is based on the architecture of VoiceFilter [3]. Note that we do not include the reference audio part in this diagram 4.7, because this loss function and training approach can be applied to any other speech separation model.

In the training process, first, the mixed audio is transformed into spectrogram using STFT and pass the model, which consists of CNN, RNN, and fully connected layer (blue box). Then the model estimates the IRM based soft mask, and it is used to reconstruct the target audio by element-wise multiplication with a mixed spectrogram. This estimated spectrogram is transformed as an audio file (WAV) after inverse STFT.

Now, our pre-trained speech recognition system Deep speech (green box) generates the transcription. This newly created transcription is used to calculate the WER with the transcription that we produced earlier using the clean audio. Again, we do not use the ground-truth transcription of Librispeech dataset to avoid the dependency of the speech recognition system. Finally, the loss is calculated with three components: clean audio spectrogram, estimated audio spectrogram, and calculated WER by Eq.4.1 (red box), and repeat the training process.

Chapter 5

Experimental Evaluation

In this chapter, we demonstrate our experiment. In the first section, we introduce our dataset with detailed information. Then the pre-processing and audio mixture generation process is following. Moreover, we present our speech recognition system to be integrated into our architecture by comparing its performance to Google speech-to-text. All the necessary software and library information are introduced later. In the second section, we compare our result with VoiceFilter model.

5.1 Experimental Setup and Data Set

5.1.1 Dataset

subset	hours	female	male	total
dev-clean	5.4	20	20	40
test-clean	5.4	20	20	40
dev-other	5.3	16	17	33
test-other	5.1	17	16	33
train-clean-100	100.6	125	126	251
train-clean-360	363.6	439	482	921
train-other-500	496.7	564	602	1166

Table 5.1: The LibriSpeech corpus dataset table from [67]

In this paper, we use LibriSpeech corpus, which is part of the LibriVox project [67]. The data contain 2,451 speakers and total 982.1hours of audio. Moreover, each audio file has its corresponding transcription, which can be used as a ground-truth text. Additionally, LibriSpeech corpus supports detailed information about the speaker such as gender as shown in Table.5.2, therefore, this can be used to generate many different types of situation.

ID	SEX	SUBSET	MINUTES	NAME
14	F	train-clean-360	25.03	Kristin LeMoine
16	F	train-clean-360	25.11	Alys AtteWater
17	M	train-clean-360	25.04	Gord Mackenzie
...

Table 5.2: Detailed information for each speaker

The combination of the mixture audio is the same as the [3]. The training dataset consists of 2,338 speakers with 281,241 combinations, and test dataset is 73 speakers with 5,567 combinations.

- Training Dataset: 281,241 (2,338 speakers), 234 hours (3 seconds each)
- Validation Dataset: 80 (80 speakers), 4 minutes (3 seconds each)
- Test Dataset: 5,567 (73 speakers), 14 hours (3-15 seconds)

The distribution of the number of words in the test dataset is shown as a histogram in Fig.5.1, and most files contain between 10 to 20 words.

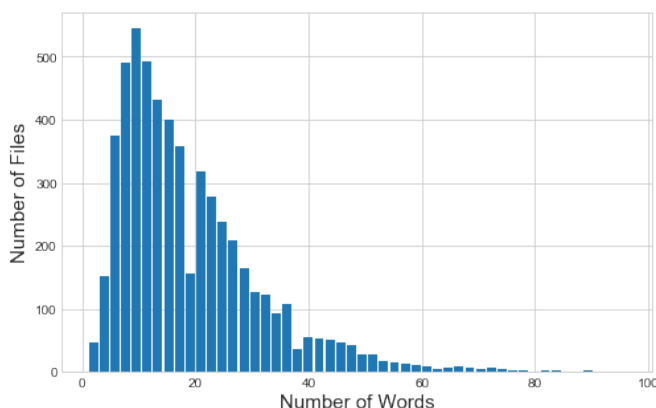


Figure 5.1: Histogram of test files by the number of words

5.1.2 Preprocessing

There are several preprocessing to enhance learning performance. First of all, all the silent part of audio files in both head and rear will be trimmed. As shown in 5.2, the head and rear part of the original audio is a silent part. If the amplitude of both edges is below a certain level, (in our case 20dB) we remove that part.

Secondly, the training audio datasets are cut into three seconds from its head part. If the original audio file is shorter than three seconds, we discard it. Even though the training

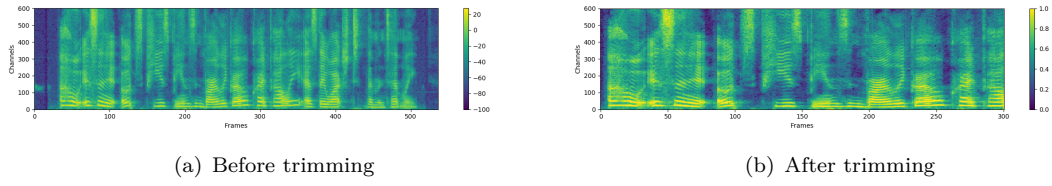


Figure 5.2: Spectrogram level difference between before and after trimming

process uses short length, the model can still inference the more extended audio since the spectrogram can be generated with the same dimension regardless of the audio file's length using phase information of mixture.

$$\hat{x}(t, f) = 1.0 + \begin{cases} 0.0 & \text{if } x(t, f)/100.0 \geq 0, \\ -1.0 & \text{if } x(t, f)/100.0 \leq -1.0, \\ \frac{x(t, f)}{100.0} & \text{otherwise.} \end{cases} \quad (5.1)$$

Thirdly, as a part of the normalization process, all the amplitude (feature) will be scaled into 0.0 to 1.0. Therefore, the gradient descent will converge faster than using the original data [68]. The scaling equation is shown in Eq.5.1, we divide the original STFT spectrogram by 100.0, and if the minimum and maximum value will be -1.0 and 0.0 respectively. The difference before and after scaling is visualized in 5.3

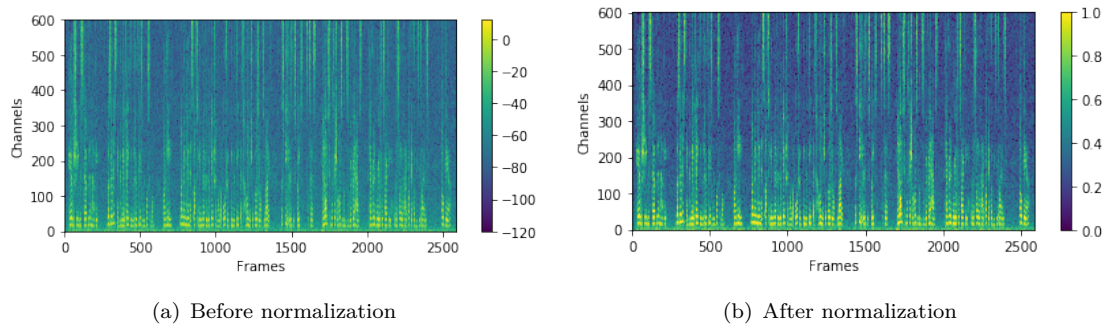


Figure 5.3: Spectrogram level difference between before and after normalization

5.1.3 Generating Audio Mixture

One way to generate the mixture dataset is manually recording the mixed and separated audio; however, this is an impractical approach. Therefore, most research produces mixed audio using direct summing of individually recorded audio, as shown in Fig.5.4. This approach can generate an infinite number of the combination. In the machine learning task, the mixed audio can be seen as input data, and the clean audio can be seen as a

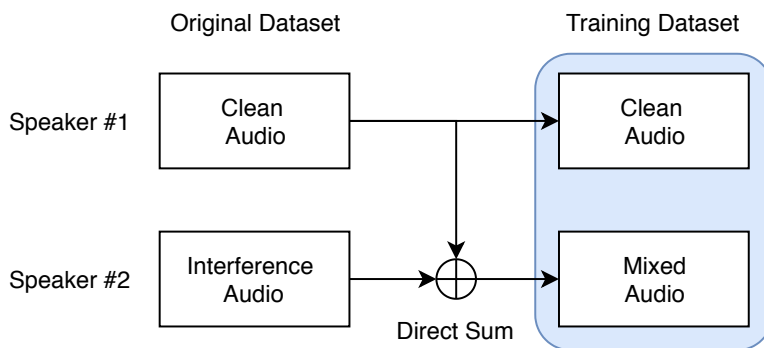


Figure 5.4: Training dataset generating process

ground-truth. Moreover, since we have speaker and gender labeled data, we can test many virtual situations.

5.1.4 Generating new ground-truth transcripts

In our system, we use a speech recognition system called DeepSpeech from [2] to calculate WER. Choosing a different speech recognition system will generate a different transcript. Therefore, we do not use the original transcript from the Librispeech corpus; we regenerate the new ground-truth text using our system. There are two benefits to this approach. First of all, as long as we are using newly generated ground-truth, the entire system is not dependent on the performance of the speech recognition system. Because, even though the system recognizes incorrectly from the original audio, it will generate a consistent result. Secondly, since we cut the original audio file into three seconds length, the original transcript is challenging to be used. We have no time-domain information in the transcripts; therefore, it is practical to generate the new text.

To validate the performance of the chosen speech recognition system, we calculate the WER from Google Speech-to-Text system which is known as the best in this area.

	Mean WER	Median WER
Deep Speech (ours)	9.6%	5.8%
Google Speech-to-Text	12.3%	9.0%

Table 5.3: Speech recognition performance comparison

We test 5,567 test dataset, and the performance is shown in table.5.3. The reason that our system is outperforming is, the model is mainly trained with Librispeech corpus. Therefore, our test dataset must be the subset of its training dataset. The model can be seen overfitted to the Librispeech corpus. However, since the main purpose of this speech recognition system is generating consistent transcripts, this overfitting is not a problem in our case. Also, the Google speech recognition system uses different syntax

model. Therefore, there is a minor word difference which has the same meaning, such as ‘Mr’ and ‘Mister’. We do not correct this difference, and this also affects the lower accuracy of the Google system.

5.1.5 Software/Hardware and Libraries Specification

Our speech separation system is based on [69]’s implementation. Therefore, *Pytorch* [70] is used as a deep learning framework. For the audio processing related task such as STFT, inverse STFT, generating audio mixture, and etc., *Librosa* [71] library is mainly used. To calculate the SDR value we use python *mir_eval* library [72]. We choose *DeepSpeech* as a speech recognition system is the implementation and pre-trained data are from [73].

Core software and libraries that are used to build our system is listed in Table.5.4.

Purpose	Title	etc.	License	Ref.
Speech Separation	VoiceFilter	PyTorch	Apache 2.0	[69]
Speech Recognition	DeepSpeech		MPL 2.0	[73]
Audio Processing	Librosa		ISC	[74]
Evaluation	mir_eval	SDR	MIT	[75]
	jiwer	WER	Apache 2.0	[76]
Web Framework	Flask		BSD 3-Clause	[77]
Miscellaneous	Numpy	tensor operation	BSD 3-Clause	[78]
	Matplotlib	plot, chart	PSF	[79]
	TensorboardX	training observation	MIT	[80]

Table 5.4: List of libraries

Note that we train our model using ‘NVIDIA Tesla V100 PCIe 32 GB’.

5.1.6 Hyperparameters and System Configuration

We introduce hyperparameters and configuration for pre-processing, training, and network itself that we use in our system in this section.

In training perspective, first of all, the batch size is 8 for the training process; therefore, each iteration eight audio files (24 seconds) are fed into the networks. Secondly, as an optimizer, Adam is used with a learning rate of 0.001. Thirdly, the checkpoint is every 1000 iteration, which means every 8000 audio files (6.6 hours). Therefore, the evaluation results (loss, WER, and SDR) are saved every 1000 iteration.

In the perspective of the network, we are following VoiceFilter’s architecture. Therefore, there is no difference between Table.4.1.

5.1.7 Training and Testing Policy

To compare our system to the original VoiceFilter, we fix training and testing policies. First of all, unlike most training process, we do not shuffle the training dataset. The primary motivation is since the dataset is over 280,000, and the training process takes a long time until it reaches the convergence, we cannot permanently wait for the best performance model. Note that, we have trained over ten days with early-stopping policy; however, the validation loss kept decreasing. Therefore, the dataset is trained in the same order for both our system and Voicefilter; then we compare both systems at every checkpoint. Secondly, since our proposed loss function is different from MSE, the comparing loss is not meaningful in our task. Therefore, the performance comparison is based on WER and SDR.

5.2 Experimental Results

In this section, we describe the result of two different systems. SDR and WER are considered as our primary measurement the same as other research. The final result is based on 80,000 iterations for both systems.

5.2.1 SDR and WER Comparison

First of all, in Fig.5.5, we compare the SDR improvement during the training process for both systems. As we mentioned earlier, since we are using the same sequence of the training dataset, we minimize the uncertainty of performance difference base on the training dataset. Therefore, as we can see from iteration 0 to around 7,000, both systems show almost the same pattern.

From the range between iteration 7,000 to 30,000, we can argue that our system is outperforming than original VoiceFilter. However, it starts to shows a similar performance with the original system between iteration 30,000 to 60,000. Then our systems show the steady pattern from iteration 70,000 to the end. Even though we cannot use this performance as our absolute evaluation since they only use 80 audio files to validate, it is crucial to confirm that our loss function is not showing abnormal behavior so that we can continue the training process.

WER improvement is also showing a similar pattern with SDR. As shown in Fig.5.6, in the early stage, both WER values are aligning on the same line (iter. 0-7,000). Then the difference starts to be seen; however, unlike the SDR pattern, it is not clear to decide which systems show better performance. However, in the range between iteration 7,000

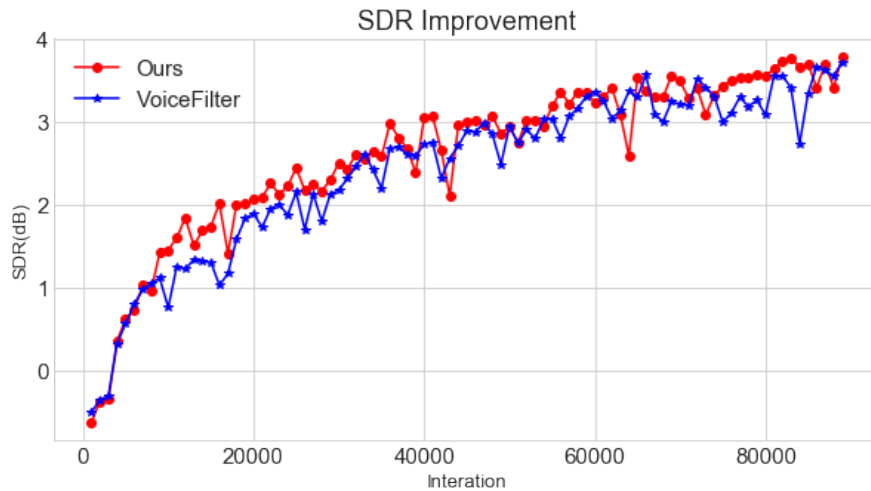


Figure 5.5: SDR improvement during training

to 30,000, we can say that our method reveals a more stable pattern, unlike the original VoiceFiter’s frequent up-down peak.

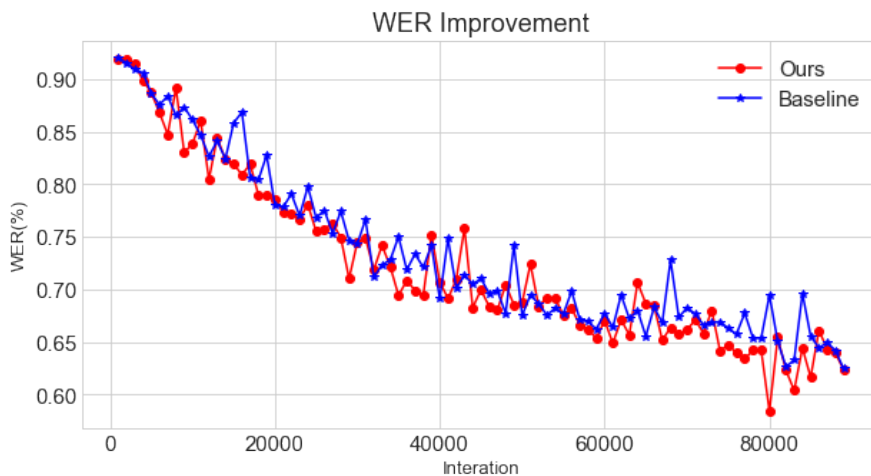


Figure 5.6: WER improvement during training

From iteration 30,000 to 60,000, WER is also showing a similar pattern with SDR; it means both system’s performance is not that different. However, after this iteration, our system shows visible lower WER (55%-65%).

Now the final performance comparison is shown in Table.5.5. First, the first row shows the mixtures speech’s SDR and WER values. Since it thoroughly mixed version, therefore, SDR values are low for both mean and median value. At the same time, WER is also showing almost 100% error rate. The last low is the upper bound using directly calculated IRM. Therefore, we call it as an Oracle system. As we can see from both SDR and WER outperforming values compared to the rest of the rows.

System	SDR(dB)		WER(%)	
	Mean	Median	Mean	Median
Before	0.10	0.13	92.7	100.0
VoiceFilter	3.99	4.85	59.2	60.0
Ours	4.09	4.41	55.7	56.1
Oracle	13.60	13.14	5.8	0.0

Table 5.5: Performance comparison

Now, if we compare the systems between ours and the original VoiceFilter, as we expect, the average WER value is improved 3.5% point, and median WER is also 3.9% point. However, our proposed approach has not enhanced only WER but also SDR evaluation than original VoiceFilter. The average SDR is 0.1(dB) improved than the original system.

5.2.2 Performance by Different Number of Words

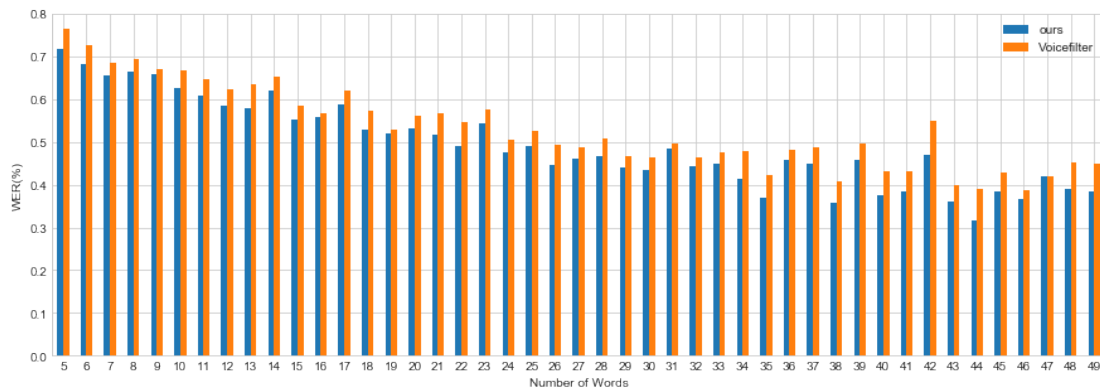


Figure 5.7: WER per different number of word in speech

In Fig.5.7, we also demonstrate the WER difference per the number of words in the audio files. Both systems show better WER when there are more words in the audio file. Notably, our model is outperforming than original VoiceFilter in all the most of the range.

The entire implementation and demo can be found in Appendix A. And, more result about the spectrogram and training process can be found in Appendix B and Appendix C respectively.

Chapter 6

Discussion

6.1 Performance Analysis

6.1.1 Evaluation Measurement Analysis

Even though the final result shows that our solution is showing better performance, however, the result is not from the converged stage; therefore, the result can be seen as preliminary. To support our achievement, we calculate the performance difference in every iteration.

First, we calculate the SDR difference between two systems and plot them in Fig.6.1.

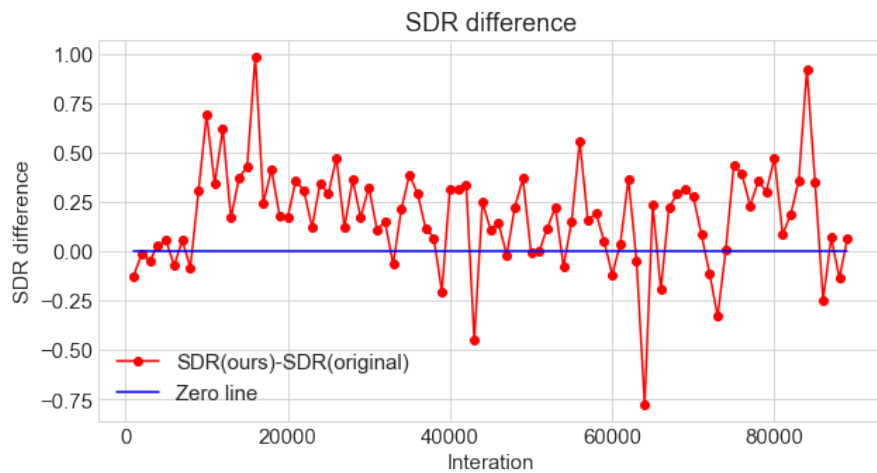


Figure 6.1: SDR difference during training

Since higher SDR means the better performance, points located on the positive side (y-axis) represent the stage that our model shows higher SDR. Out of 89 checkpoints, there are 69 points that the proposed model is outperforming. Moreover, if we average the

difference for each case, then the evidence is more clear. The average SDR difference when our model is performing better is 0.273, while the other case is -0.15 ($|0.273| > |-0.156|$).

WER is also showing a similar pattern shown in Fig.6.2. In WER comparison, lower value means better performance; therefore, the graph shows more negative points. Our model performs better in 62 checkpoints while the original model is outperforming only 27 points while the average difference is 0.013 and -0.021 ($|0.013| < |-0.021|$).

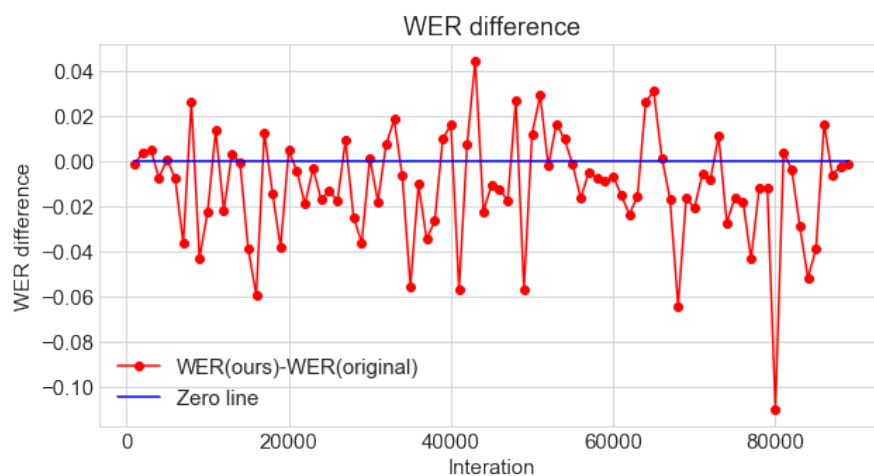


Figure 6.2: WER difference during training

6.1.2 Relationship between SDR and WER

One question we keep having during the project is, “*Is the improvement of SDR highly relevant to the performance of WER?*”. In theory, its higher SDR means lower noisy and interfere source; therefore, it is reasonable to assume that if we achieve the higher SDR, the WER will relatively increase. Therefore we calculate the correlation coefficient between SDR improvement and WER improvement for each iteration.

Measurement	Correlation coefficient	
	Ours	VoiceFilter
SDR vs. Loss	-0.988	-0.986
WER vs. Loss	0.960	0.956
SDR vs. WER	-0.967	-0.970

Table 6.1: The correlation coefficients between the three measurements during the training for validation dataset

As we can see from Table.6.1, there is a small difference between two systems, however, since both coefficients are close to $|\text{coef}| = 1$, it is not considerable. WER and SDR are also highly correlated; therefore, we may think the training approach that improving the signal-level performance will improve the transcription-level performance. However,

since they are an average of 80 datasets, and regardless of each audio’s original SDR, we only measure the improved SDR, it is not strong evidence. Moreover, we also have observed some case that shows better WER while SDR decreased. Therefore this time we calculate the correlation coefficient based on individual improvement as shown in Tab.6.2

Measurement	Correlation coefficient	
	Ours	VoiceFilter
SDR imprv. vs. WER imprv.	-0.046	-0.032

Table 6.2: The correlation coefficients between improvement SDR and WER for each audio

As we can see, both coefficients show almost 0 correlation; in other words, even though SDR improves, it does not necessarily guarantee better WER.

We do not investigate further for this discovery; however, this can be interesting future work for choosing the appropriate signal-level measurement for the transcription-level measurement.

6.2 Limitations

6.2.1 Dataset

There is one limitation to compare the performance to other research directly. This is why we cannot offer comparable result to the other speech separation research. In the original paper of VoiceFilter [3], the SDR and WER that they state are 17.9dB, 23.4% respectively, and it has a big difference with our result (4.09dB and 55.7%). However, the original paper’s test dataset already has high SDR and low WER values (10.1db and 55.9%) even before the separation. First time when we could not reach the similar performance we suspect our foundation code or pre-processing, however when we calculate the SDR and WER value (before separation) with the test dataset that they published we could not get the same result as they published. Therefore, we contacted the original author Quan Wang, and we are confirmed that the final result is calculated with *a big batch of three seconds segments*, and it can have some randomness. The way they generate the final test set is illustrated in Fig.6.3

For example, if the clean audio is 15 seconds, and the noise audio is 5 seconds, rest 10 seconds is technically the clean audio. Then, if we segment using 3 seconds windows (red box), the rear part of the test data is still clean audio. This is why their performance before the separation is already high. Therefore, it is not possible to generate the

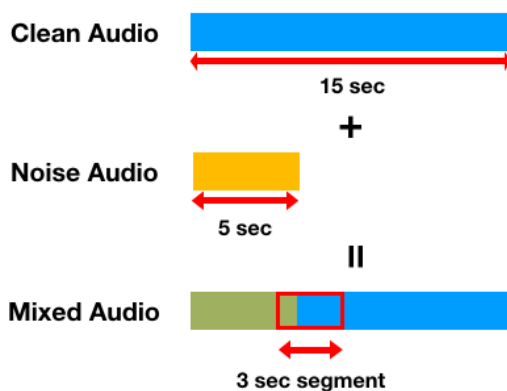


Figure 6.3: Sampling for evaluation

same mixture that they used; likewise, it is difficult to compare to the state-of-the-art result since the segmentation information is not public. We could have experimented with using other datasets such as WSJ0mix; however, this dataset is not open source (membership-based).

6.2.2 Training Time

As we mentioned above, our system requires longer training time than conventional end-to-end speech separation system. Therefore, it is challenging to train with many different hyperparameter setting. Moreover, even though we trained for 12 days, we could not reach the convergence of our model performance. Fig.6.4, shows the difference



Figure 6.4: Training time comparison

in the training process in time-wise. To train the same number of iteration, the original VoiceFilter needs one day (blue line), unlike our model (red line), which requires ten days. Even though the training time is not the main performance criteria for the machine learning process since once a model is trained the inferencing time is not different, this time consumption problem should be improved for the more diverse training approach.

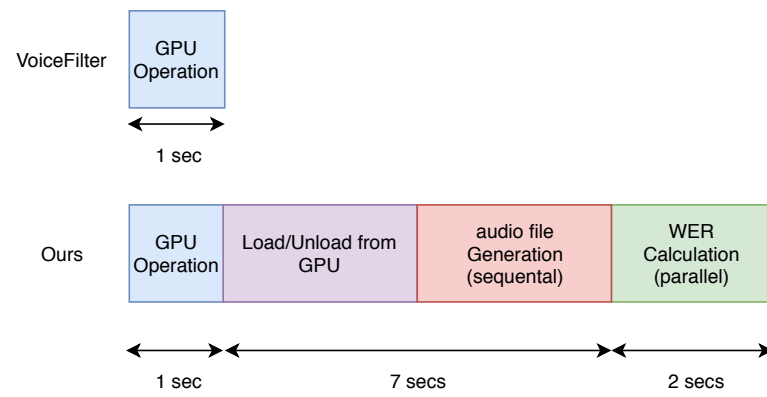


Figure 6.5: Training time comparison per iteration

Fig.6.5 shows the time-consumption per iteration. The main consuming part is when we generate the audio file from the spectrogram. Unlike the WER calculation is operating parallelly, the generation part is a sequential operation; therefore this process needs to be improved. For example, if we can calculate the WER without generating the audio file, the process can be much faster.

Chapter 7

Conclusion and Future Directions

In this thesis, we have introduced a new approach to training for the speech separation task. There are two main contributions to enhance the existing speech separation model.

First of all, we have proposed the new loss function for the separation task, which can associate both the signal-level and transcription-level performance measurement directly. Therefore, we have improved WER from 59.7% to 55.7%; moreover, even SDR is elevated from 3.99dB to 4.09dB compared to the existing model. This result proves that applying WER to loss-function can improve the separation performance from both signal and transcription perspective. Secondly, our generalized integration architecture can be used for any other systems without being constrained by the system environment. Therefore, we can propose a more diversified combination of speech separation and recognition systems.

This experiment can be improved by having several adjustments. Firstly, we can train on bigger batch size so that we can shorten the training time and try a more diverse experiment. Secondly, combining different systems such as PIT and DPCL with Google speech-to-text and Amazon Transcribe can prove our approach more strongly. Lastly, investigating the relationship between word length and performance can also be an interesting topic.

List of Figures

2.1	Subdomains of ASR	5
2.2	Audio wave plot	7
2.3	Frequency plot	7
2.4	Amplitude-Time plot	8
2.5	Amplitude-Frequency plot	8
2.6	Spectrogram after STFT of audio sources	9
2.7	Signal separation in the spectrogram	10
2.8	IBM Spectrogram	11
2.9	IRM Spectrogram	11
2.10	Sample graph between blood pressure and weight	14
2.11	Sample graph between heart attack and weight	15
2.12	Sigmoid Function	15
2.13	Gradient descent simple example	17
2.14	Neural network cell with activation function	20
2.15	One hidden layer neural networks	20
2.16	Two hidden layer neural networks	21
2.17	Basic architectures of artificial neural networks and deep neural networks	22
2.18	Different types of activation function	25
2.19	Recurrent neural networks	25
2.20	LSTM cell architecture	27
2.21	Example of CNNs architecture	28
2.22	Convolutional process on 4x4 image using 3x3 kernel	29
2.23	Padding	29
2.24	Stride	30
2.25	Depth	30
2.26	Dilation	31
2.27	Example of pooling process	32
2.28	Example of vectorizing the words	32
2.29	One-hot encoding	33
2.30	Dense representation	33
2.31	Simple neural network for image recognition task	34
3.1	Source/mask estimation architecture to separate the original source	36
3.2	DPCL system architecture.	39
3.3	PCA result of speech embedding space, figure from [42]	40
3.4	DANet System Architecture	41
3.5	PIT System Architecture	42

3.6	SDR improvements for different speech separation system using WSJ0-2MIX dataset, the figure from [6]	42
3.7	Joint training architecture using both speech separation and speech recognition, the figure from [48]	43
3.8	Diagram of reweighting algorithm, figure from [54]	46
4.1	VoiceFilter System Architecture from [3]	50
4.2	The location of d -vector in the neural networks	51
4.3	Time and frequency focused kernel size	52
4.4	Layer-level training process	53
4.5	Simultaneous request for speech recognition	55
4.6	Exchangeable architecture with other systems	56
4.7	System Architecture	56
5.1	Histogram of test files by the number of words	60
5.2	Spectrogram level difference between before and after trimming	61
5.3	Spectrogram level difference between before and after normalization	61
5.4	Training dataset generating process	62
5.5	SDR improvement during training	65
5.6	WER improvement during training	65
5.7	WER per different number of word in speech	66
6.1	SDR difference during training	67
6.2	WER difference during training	68
6.3	Sampling for evaluation	70
6.4	Training time comparison	70
6.5	Training time comparison per iteration	71
A.1	Speech separation demo generated by our model	79
B.1	Mixture Spectrogram	81
B.2	Target Spectrogram	81
B.3	Separated Spectrogram	81
B.4	Estimated Mask Spectrogram	82
B.5	Error-difference Spectrogram	82
C.1	SDR improvement during training	83
C.2	WER improvement during training	83
C.3	Validation loss during training	84
C.4	Training loss during training	84
C.5	Training WER during training	84

List of Tables

4.1	Parameters of the VoiceFilter, table from [3]	52
5.1	The LibriSpeech corpus dataset table from [67]	59
5.2	Detailed information for each speaker	60
5.3	Speech recognition performance comparison	62
5.4	List of libraries	63
5.5	Performance comparison	66
6.1	The correlation coefficients between the three measurements during the training for validation dataset	68
6.2	The correlation coefficients between improvement SDR and WER for each audio	69

Appendix A

Implementation and Demo

The implementation of our system can be found in our Github repository: <https://github.com/thejungwon/mwetss>.

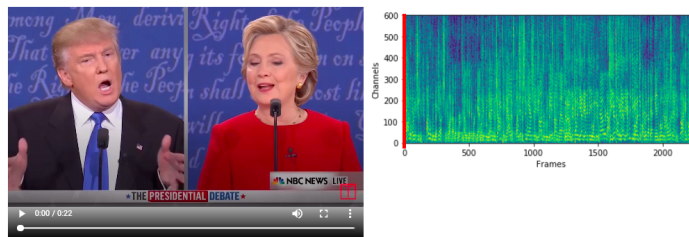
The detailed installation process is described in README.md. The speech recognition implementation is based on [73], and the speech separation code is based on [69].

Our demo is available in: <https://master-thesis-a24e7.firebaseio.com/>.

Minimum Word Error Rate Training for Speech Separation System

by Jungwon Seo

Try the Demo!



Trump Voice Only



Hillary Voice Only



Figure A.1: Speech separation demo generated by our model

Appendix B

Spectrogram Results

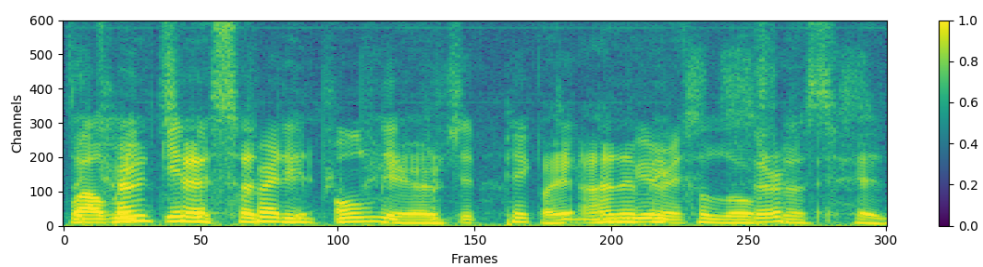


Figure B.1: Mixture Spectrogram

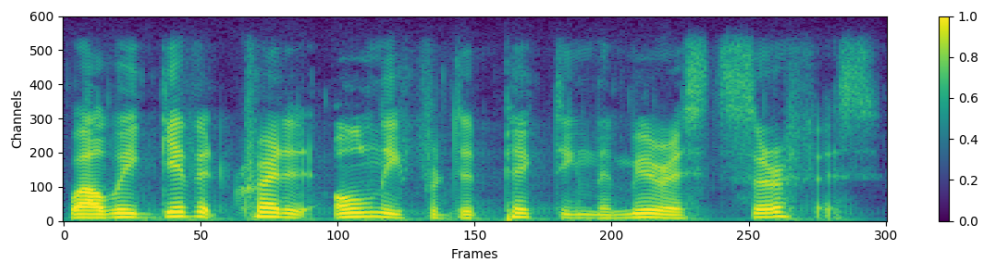


Figure B.2: Target Spectrogram

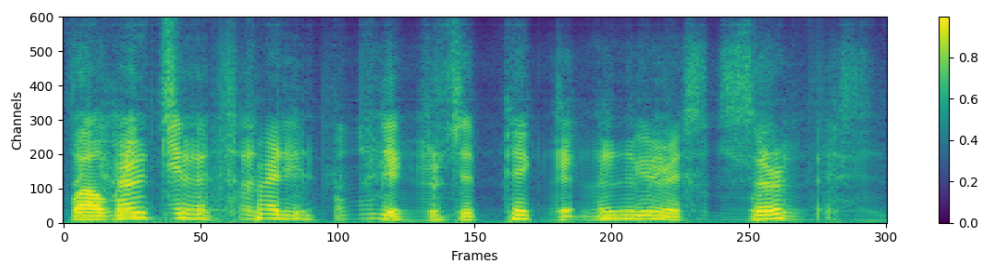


Figure B.3: Separated Spectrogram

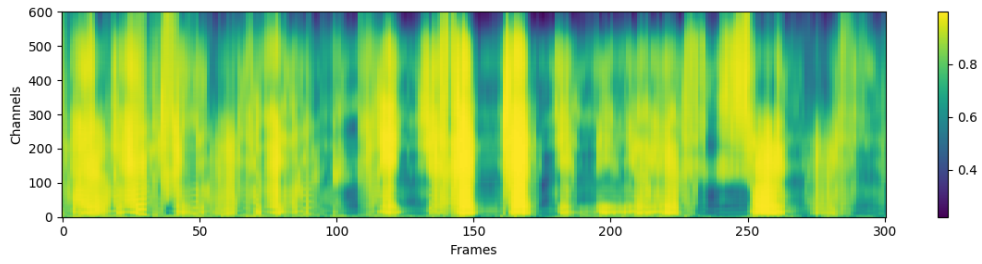


Figure B.4: Estimated Mask Spectrogram

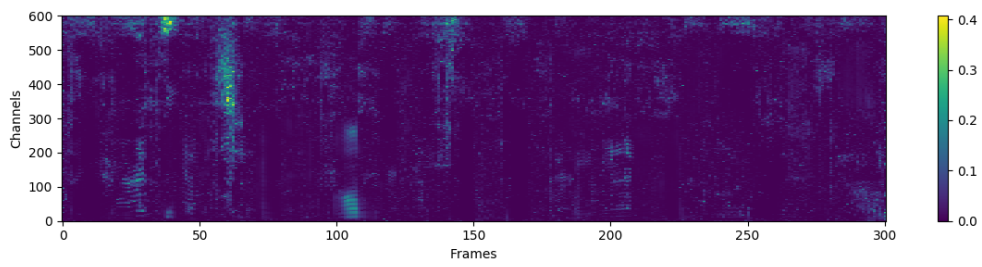


Figure B.5: Error-difference Spectrogram

Appendix C

Training Process

The red line indicates our model, while the pink line shows the original model.

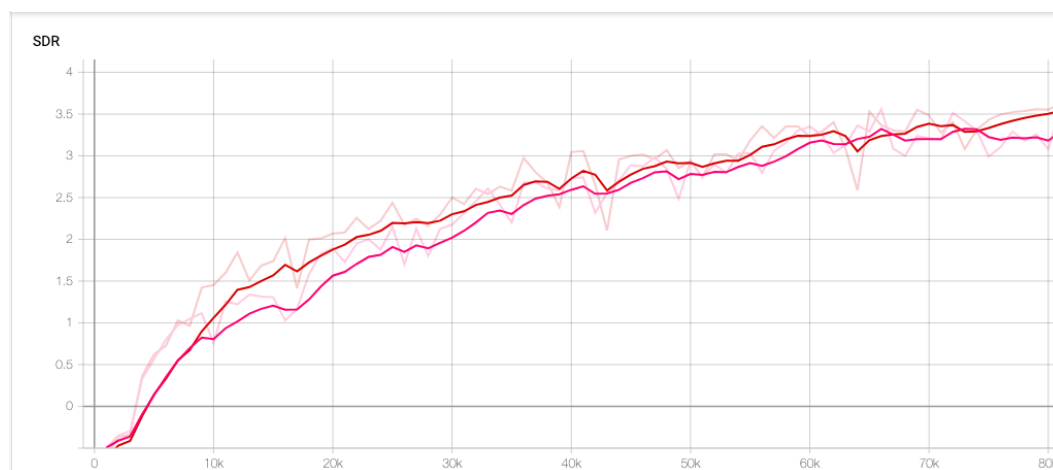


Figure C.1: SDR improvement during training

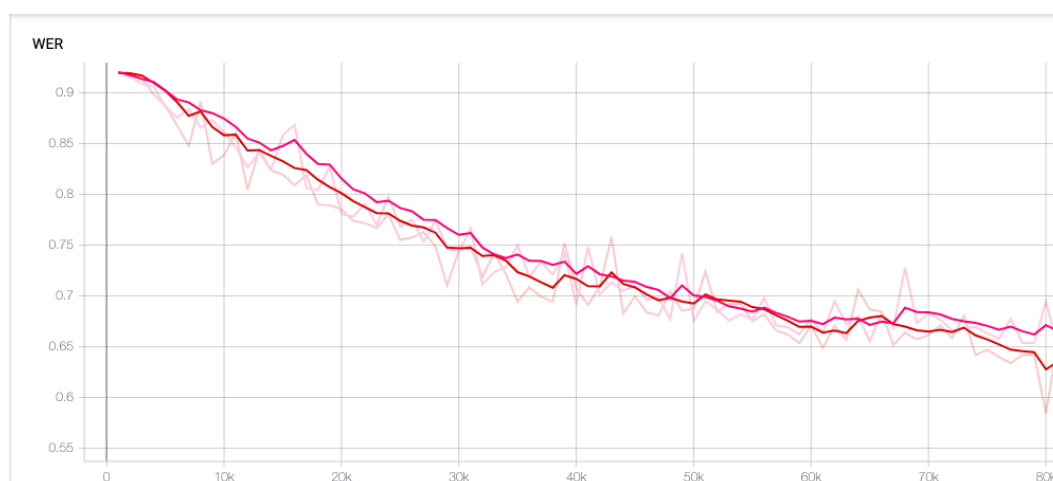


Figure C.2: WER improvement during training

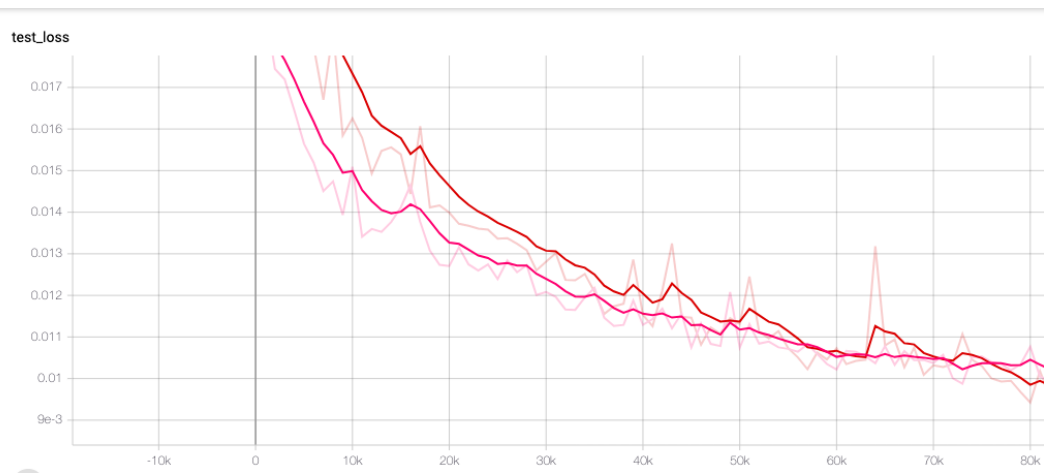


Figure C.3: Validation loss during training



Figure C.4: Training loss during training

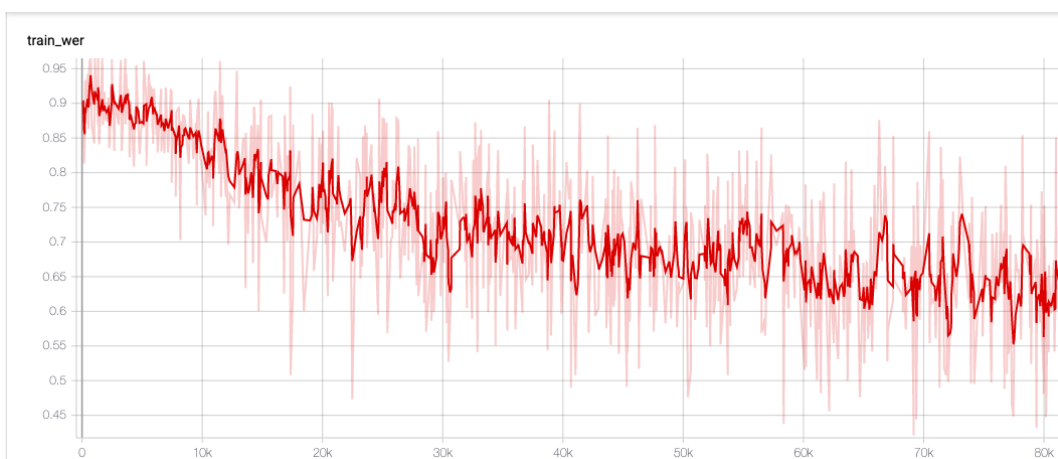


Figure C.5: Training WER during training

Bibliography

- [1] DeLiang Wang and Jitong Chen. Supervised speech separation based on deep learning: An overview. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(10):1702–1726, 2018.
- [2] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.
- [3] Quan Wang, Hannah Muckenhirn, Kevin Wilson, Prashant Sridhar, Zelin Wu, John Hershey, Rif A Saurous, Ron J Weiss, Ye Jia, and Ignacio Lopez Moreno. Voicefilter: Targeted voice separation by speaker-conditioned spectrogram masking. *arXiv preprint arXiv:1810.04826*, 2018.
- [4] E Colin Cherry. Some experiments on the recognition of speech, with one and with two ears. *The Journal of the acoustical society of America*, 25(5):975–979, 1953.
- [5] Nima Mesgarani and Edward F Chang. Selective cortical representation of attended speaker in multi-talker speech perception. *Nature*, 485(7397):233, 2012.
- [6] Yan-min Qian, Chao Weng, Xuan-kai Chang, Shuai Wang, and Dong Yu. Past review, current progress, and challenges ahead on the cocktail party problem. *Frontiers of Information Technology & Electronic Engineering*, 19(1):40–63, 2018.
- [7] John R Hershey, Steven J Rennie, Peder A Olsen, and Trausti T Kristjansson. Super-human multi-talker speech recognition: A graphical modeling approach. *Computer Speech & Language*, 24(1):45–66, 2010.
- [8] Ervin Sejdić, Igor Djurović, and Jin Jiang. Time–frequency feature representation using energy concentration: An overview of recent advances. *Digital signal processing*, 19(1):153–183, 2009.
- [9] Jonathan Allen. Short term spectral analysis, synthesis, and modification by discrete fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(3):235–238, 1977.

-
- [10] Brian ED Kingsbury, Nelson Morgan, and Steven Greenberg. Robust speech recognition using the modulation spectrogram. *Speech communication*, 25(1-3): 117–132, 1998.
- [11] Promod Yenigalla, Abhay Kumar, Suraj Tripathi, Chirag Singh, Sibsambhu Kar, and Jithendra Vepa. Speech emotion recognition using spectrogram & phoneme embedding. *Proc. Interspeech 2018*, pages 3688–3692, 2018.
- [12] Sanjay Krishna Gouda, Salil Kanetkar, David Harrison, and Manfred K Warmuth. Speech recognition: Keyword spotting through image recognition. *arXiv preprint arXiv:1803.03759*, 2018.
- [13] John R Hershey, Zhuo Chen, Jonathan Le Roux, and Shinji Watanabe. Deep clustering: Discriminative embeddings for segmentation and separation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 31–35. IEEE, 2016.
- [14] Bhiksha Raj and Richard M Stern. Missing-feature approaches in speech recognition. *IEEE Signal Processing Magazine*, 22(5):101–116, 2005.
- [15] Jon Barker, Ljubomir Josifovski, Martin Cooke, and Phil Green. Soft decisions in missing data techniques for robust automatic speech recognition. In *Sixth International Conference on Spoken Language Processing*, 2000.
- [16] Arun Narayanan and DeLiang Wang. Ideal ratio mask estimation using deep neural networks for robust speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7092–7096. IEEE, 2013.
- [17] Yuxuan Wang, Arun Narayanan, and DeLiang Wang. On training targets for supervised speech separation. *IEEE/ACM transactions on audio, speech, and language processing*, 22(12):1849–1858, 2014.
- [18] Hakan Erdogan, John R Hershey, Shinji Watanabe, and Jonathan Le Roux. Phase-sensitive and recognition-boosted speech separation using deep recurrent neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 708–712. IEEE, 2015.
- [19] Hakan Erdogan, John R Hershey, Shinji Watanabe, and Jonathan Le Roux. Deep recurrent networks for separation and recognition of single-channel speech in non-stationary background audio. In *New Era for Robust Speech Recognition*, pages 165–186. Springer, 2017.
- [20] Soundararajan Srinivasan, Nicoleta Roman, and DeLiang Wang. Binary and ratio time-frequency masks for robust speech recognition. *Speech Communication*, 48(11): 1486–1501, 2006.

-
- [21] Peter Assmann and Quentin Summerfield. The perception of speech under adverse conditions. In *Speech processing in the auditory system*, pages 231–308. Springer, 2004.
- [22] Julien Meyer, Laure Dentel, and Fanny Meunier. Speech recognition in natural background noise. *PloS one*, 8(11):e79279, 2013.
- [23] Xu Li, Junfeng Li, and Yonghong Yan. Ideal ratio mask estimation using deep neural networks for monaural speech segregation in noisy reverberant conditions. In *Interspeech*, pages 1203–1207, 2017.
- [24] Morten Kolbæk, Dong Yu, Zheng-Hua Tan, Jesper Jensen, Morten Kolbaek, Dong Yu, Zheng-Hua Tan, and Jesper Jensen. Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 25(10):1901–1913, 2017.
- [25] Emmanuel Vincent, Rémi Gribonval, and Cédric Févotte. Performance measurement in blind audio source separation. *IEEE transactions on audio, speech, and language processing*, 14(4):1462–1469, 2006.
- [26] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.
- [27] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [28] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [29] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.
- [30] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- [31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [32] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

- [33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [34] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.
- [35] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.
- [36] Tom Sercu and Vaibhava Goel. Dense prediction on sequences with time-dilated convolutions for speech recognition. *arXiv preprint arXiv:1611.09288*, 2016.
- [37] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [38] Guy J Brown and Martin Cooke. Computational auditory scene analysis. *Computer Speech & Language*, 8(4):297–336, 1994.
- [39] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.
- [40] Zoubin Ghahramani and Michael I Jordan. Factorial hidden markov models. In *Advances in Neural Information Processing Systems*, pages 472–478, 1996.
- [41] Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis. Joint optimization of masks and deep recurrent neural networks for monaural source separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(12):2136–2147, 2015.
- [42] Zhuo Chen, Yi Luo, and Nima Mesgarani. Deep attractor network for single-microphone speaker separation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 246–250. IEEE, 2017.
- [43] Patricia K Kuhl. Human adults and human infants show a perceptual magnet effect for the prototypes of speech categories, monkeys do not. *Perception & psychophysics*, 50(2):93–107, 1991.
- [44] Morten Kolbæk, Dong Yu, Zheng-Hua Tan, and Jesper Jensen. Joint separation and denoising of noisy multi-talker speech using recurrent neural networks and permutation invariant training. In *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2017.

- [45] Yanmin Qian, Xuankai Chang, and Dong Yu. Single-channel multi-talker speech recognition with permutation invariant training. *Speech Communication*, 104:1–11, 2018.
- [46] Xuankai Chang, Yanmin Qian, and Dong Yu. Adaptive permutation invariant training with auxiliary information for monaural multi-talker speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5974–5978. IEEE, 2018.
- [47] Soyeon Choe, Soo-Whan Chung, Youna Ji, and Hong-Goo Kang. Orthonormal embedding-based deep clustering for single-channel speech separation. *arXiv preprint arXiv:1901.04690*, 2019.
- [48] Shane Settle, Jonathan Le Roux, Takaaki Hori, Shinji Watanabe, and John R Hershey. End-to-end multi-speaker speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4819–4823. IEEE, 2018.
- [49] Yi Luo, Zhuo Chen, John R Hershey, Jonathan Le Roux, and Nima Mesgarani. Deep clustering and conventional networks for music separation: Stronger together. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 61–65. IEEE, 2017.
- [50] Suyoun Kim, Takaaki Hori, and Shinji Watanabe. Joint ctc-attention based end-to-end speech recognition using multi-task learning. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4835–4839. IEEE, 2017.
- [51] Zhong-Qiu Wang, Jonathan Le Roux, and John R Hershey. Alternative objective functions for deep clustering. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 686–690. IEEE, 2018.
- [52] Rohit Prabhavalkar, Tara N Sainath, Yonghui Wu, Patrick Nguyen, Zhifeng Chen, Chung-Cheng Chiu, and Anjuli Kannan. Minimum word error rate training for attention-based sequence-to-sequence models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4839–4843. IEEE, 2018.
- [53] Mirjam Killer, Sebastian Stuker, and Tanja Schultz. Grapheme based speech recognition. In *Eighth European Conference on Speech Communication and Technology*, 2003.
- [54] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. *arXiv preprint arXiv:1803.09050*, 2018.

- [55] Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. Generalized end-to-end loss for speaker verification. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4879–4883. IEEE, 2018.
- [56] Georg Heigold, Ignacio Moreno, Samy Bengio, and Noam Shazeer. End-to-end text-dependent speaker verification. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5115–5119. IEEE, 2016.
- [57] Yu-hsin Chen, Ignacio Lopez-Moreno, Tara N Sainath, Mirkó Visontai, Raziel Alvarez, and Carolina Parada. Locally-connected and convolutional neural networks for small footprint speaker recognition. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.
- [58] Ehsan Variani, Xin Lei, Erik McDermott, Ignacio Lopez Moreno, and Javier Gonzalez-Dominguez. Deep neural networks for small footprint text-dependent speaker verification. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4052–4056. IEEE, 2014.
- [59] Kevin Wilson, Michael Chinen, Jeremy Thorpe, Brian Patton, John Hershey, Rif A Saurous, Jan Skoglund, and Richard F Lyon. Exploring tradeoffs in models for low-latency speech enhancement. In *2018 16th International Workshop on Acoustic Signal Enhancement (IWAENC)*, pages 366–370. IEEE, 2018.
- [60] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.
- [61] Byron C Wallace, Kevin Small, Carla E Brodley, and Thomas A Trikalinos. Class imbalance, redux. In *2011 IEEE 11th international conference on data mining*, pages 754–763. IEEE, 2011.
- [62] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*, volume 97, pages 179–186. Nashville, USA, 1997.
- [63] Gilles Cohen, Mélanie Hilario, Hugo Sax, Stéphane Hugonnet, and Antoine Geissbuhler. Learning from imbalanced data in surveillance of nosocomial infection. *Artificial intelligence in medicine*, 37(1):7–18, 2006.
- [64] Xingye Qiao and Yufeng Liu. Adaptive weighted learning for unbalanced multicategory classification. *Biometrics*, 65(1):159–168, 2009.
- [65] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3067–3075. JMLR. org, 2017.

- [66] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000.
- [67] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210. IEEE, 2015.
- [68] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [69] Seungwon Park. Voicefilter. <https://github.com/mindslab-ai/voicefilter>, 2019.
- [70] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
- [71] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, pages 18–25, 2015.
- [72] Colin Raffel, Brian McFee, Eric J Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, Daniel PW Ellis, and C Colin Raffel. mir_eval: A transparent implementation of common mir metrics. In *In Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR*. Citeseer, 2014.
- [73] Mozilla. Deepspeech. <https://github.com/mozilla/DeepSpeech>, 2016.
- [74] Brian McFee and Colin Raffel. Librosa. <https://github.com/librosa/librosa>, 2015.
- [75] Colin Raffel. mir_eval. https://github.com/craffel/mir_eval, 2014.
- [76] Nik Vaessen. jiwer. <https://github.com/jitsi/asr-wer>, 2018.
- [77] Armin Ronacher. Flask. <https://github.com/pallets/flask>, 2010.
- [78] Travis Oliphant. Numpy. <https://github.com/numpy/numpy>, 2006.
- [79] John D. Hunter. Matplotlib. <https://github.com/matplotlib/matplotlib>, 2003.
- [80] Tzu-Wei Huang. Tensorboard x. <https://github.com/lanpa/tensorboardX>, 2017.